1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses
on Microfiche Service

Services des thèses canadiennes
sur microfiches

Ottawa, Canada
K1A 0N4

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Ecrire en lettres moulees ou dactylographier

Full Name of Author — Nom complet de l'auteur

David Duncan Ridewood Glendinning

Date of Birth — Date de naissance

1985 10 23

Country of Birth — Lieu de naissance

Canada

Permanent Address — Résidence fixe

21 Tamblyn Crescent, Kanata, Ontario, K2L 3A5

Title of Thesis — Titre de la thèse

A Personal Workstation Based CAD Environment for Electrical Engineering Education

University — Université

Carleton University

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

M.Eng.

Year this degree conferred — Année d'obtention de ce grade

1985

Name of Supervisor — Nom du directeur de thèse

Dr. M.A. Copeland

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication, ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

1985 09 20

Signature

David Ridewood

**A Personal Workstation Based CAD Environment
for Electrical Engineering Education**

by

D. Duncan R. Glendinning, B. Eng.

**A thesis submitted to the Faculty of Graduate
Studies and Research in partial fulfillment
of the requirements for the degree of**

Master of Engineering

**Department of Electronics
Faculty of Engineering
Carleton University
© Ottawa, Canada
August 1985**

The undersigned recommend to the Faculty of Graduate Studies and Research the acceptance of the thesis: "A Personal Workstation Based CAD Environment for Electrical Engineering Education" submitted by D. Duncan R. Glendinning in partial fulfilment of the requirements for the degree of Master of Engineering.

Prof. M.A. Copeland
Thesis Supervisor

Prof. A.R. Boothroyd
Chairman
Department of Electronics

August, 1985

Abstract

With the advent of inexpensive computers, it has become possible to integrate these machines into electrical engineering education and research in the form of computer aided design (CAD) workstations.

This thesis documents the software development of an inexpensive engineering workstation environment suitable for education in the analysis and design of semiconductor integrated circuits. The design information from this workstation environment can lead to fabrication, either in the Department's IC facilities, or via the Canadian Microelectronics Corporation.

in disk storage savings.

To share a file system amongst many users requires a network to connect all users and file systems together. This leads to adopting a networked microprocessor. The choice adopted made the Department of Electronics among the first to do so in a low cost teaching environment. The two popular network technologies are RS422 (a twisted pair base band scheme operating at 1M Bit per sec.) and Ethernet (a coax base band system operating at either 3 or 10 MBPS). Despite the obvious speed advantage that is offered by Ethernet, its cost is high, and it is not available as an option for all machines. RS422 is inexpensive in that it can use quad telephone cable as its network medium, and the network interface is quite often built into machines which are useable in a network (Corvus, Quark, DY4).

At the time of the beginning of this work, the Corvus Concept was clearly the most appropriate choice given the previous discussion. Since that time, more contemporary machines have become available, as will be discussed in section 2.5.

Operating in a network environment provides the advantage of amortizing costly peripherals (plotters, printers, disk drives) over all user nodes. The alternatives are to equip every node with the peripherals, or to establish a complicated switching system - a "local area network for peripherals". The disadvantage of a networked system is that the hard disk presents a critical point failure for the system.

2.3 Software Support for Program Development

Besides providing file manipulation and operating systems software, if these machines are to be used in a engineering CAD environment, a graphics kernel would have to be available in order to convert/develop an IC layout program. This allows high level language program implementation using standard graphics calls rather than having the application program directly control the graphics hardware. Such kernels usually support more than one graphics device. They manage all the world-to-viewport-to-device coordinate space transformations. Below the graphics kernel lies the device driver, which is responsible for formatting the data in a way which is compatible with the currently selected graphics device.

If one expected the simulation results to agree with those on other machines, floating point calculations software would have to be performed according to the IEEE floating point standard.

Table of Contents

	Page
Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Figures	v
List of Appendices	vi
 Chapter 1 - Introduction	
1.0 Thesis Objective	1
1.1 Motivation	1
1.2 Background	3
1.3 Thesis Format	5
 Chapter 2 - Hardware Requirements for the VLSI Design Environment	
2.0 Introduction	6
2.1 Processor Choice	6
2.1.1 The Intel 8086	6
2.1.2 The Motorola 68000	7
2.2 Distributed versus Centralized Computing	8
2.3 Software Support for Program Development	9
2.4 The Corvus Concept	10
2.5 Currently Available Machines	11
 Chapter 3 - Software Requirements for the VLSI Design Environment	
3.0 Introduction	13
3.1 Two-Dimensional Graphics Library	14
3.1.0 Introduction	14
3.1.1 Role of Graphics Package	14
3.1.2 User Interface	15
3.1.2.1 Type Declarations	17
3.1.3 Interface Definitions	18
3.1.4 Implementation Details	22
3.1.4.1 Device Driver Interface Specifications	22
3.1.4.2 Software Character Definition	25
3.1.5 Device Driver Implementation Details	25
3.1.5.1 Vector to Raster Scan Conversion	25
3.1.5.2 Line Drawing Algorithm	27
3.2 IC Layout	31
3.2.0 Introduction	31
3.2.1 CifDraw Description	31
3.2.2 CifDraw Implementation	36
3.3 Circuit Simulation	42
3.4 Logic Simulation	44
3.5 Filter Analysis	47
3.5.0 Introduction	47
3.5.1 The Complex Program	47

Chapter 4 - Design Environment Utilization	
4.0 Introduction	51
4.1 User Accounts	51
4.2 Network Consolidation	52
4.3 The Network	53
4.4 Educational Use	53
4.4.1 Graduate Use	53
4.4.2 Undergraduate Use	55
Chapter 5 - Conclusions	
5.1 Summary	57
5.2 Future Work	58
5.2.0 Introduction	58
5.2.1 Hardware Improvements	58
5.2.2 Software Improvements	60
5.2.3 Future Machines	61
5.3 Conclusions	62
References	63
Bibliography	64
Appendices	65
Appendix A: Two-Dimensional Graphics Library Software Listings	67
A-1: GraphDefs Unit	68
A-2: Graph Unit	69
A-3: IoConcept Unit	82
A-4: IoLa120 Unit	85
A-5: IoHP7221 Unit	93
A-6: IoWindow Unit	103
A-7: Create Program	104
Appendix B: CifDraw IC Layout Program Software Listings	109
B-1: LayoutDefs Unit	110
B-2: CifDraw Program	113
B-3: CifBias Program	149
Appendix C: COMPLEX Filter Analysis Program Software Listings	154
C-1: Complex Program	155
C-2: Plot Unit	166
C-3: Export Program	171
Appendix D: Handbook As Seen By Students	175

List of Figures

Figure Number	Description	Page
1	Conventional Integrated Circuit Design Process	1
2	Space Coordinate Transformations	14
3	Graphics System Routine Organization	16
4	Vector to Raster Scan Conversion Process	27
5	Memory Image of Overlapping Data Structures	28
6	Resultant Rasterized Image	30
7	CifDraw Video Screen	32
8	CifDraw Internal Data Structures	37
9	Unmodified Internal Coordinate Space	38
10	Modified Internal Coordinate Space	38
11	Nine Transistor CMOS Operational Amplifier	43
12	Data Flow for SPLICE Simulations	45
13	Boot Strapped Inverter	46
14	CAD Network	57

List of Appendices

Appendix Number	Description	Page
A-1	Two-Dimensional Graphics Library Global Declarations	68
A-2	Two-Dimensional Graphics Library Program Listings	69
A-3	IoConcept Device Driver Software Listing	82
A-4	IoLA120 Device Driver Software Listing	85
A-5	IoHP7221 Device Driver Software Listing	93
A-6	IoWindow Device Driver Software Listing	103
A-7	Software Character Definition Program Listing	104
B-1	CifDraw Program Global Declarations	110
B-2	CifDraw Program Listing	113
B-3	CifBias Program Listing	149
C-1	COMPLEX Program Listing	155
C-2	COMPLEX Graphics Support Listing	166
C-3	Export File Transfer Program Listing	171
D	Handbook As Seen By Students	175

Chapter 1 - Introduction

1.0 Thesis Objective

The goal of this thesis is to produce an environment suitable for teaching computer-based integrated circuit design of large-scale integrated circuits at the university level. This work diverges from industrial computer aided design (CAD) workstations in that low cost is crucial.

1.1 Motivation

The design and implementation of an integrated circuit (IC) in the absence of automated design tools is iterative in nature, and can take weeks to months between revisions depending upon the design tools available. A designer rarely produces a 100% working circuit on the first iteration. The following diagram illustrates the iterative nature of this manual design process for producing an IC from the circuit idea, assuming that a device level circuit simulator and an interactive IC layout editor are available:

Figure 1 - Conventional Integrated Circuit Design Process

Not having these basic tools (IC layout editor and circuit simulator) readily available and easily accessible to students precludes offering IC design courses in large to very large scale integration (VLSI) at the university level. Research oriented circuit design is also limited in extent and complexity if these basic tools are not available.

Mead and Conway[1] had a revolutionary effect on IC design education by providing a

book entitled "Introduction to VLSI Systems", which documented a structured design methodology. Their hierarchical design approach employs two-phase non-overlapping clocked dynamic logic to construct synchronous circuits, which can be fabricated in an nMOS process. They describe methods of using regularized structures (like registers) in a hierarchical subsystem design (like a register file), to reduce circuit complexity, often with reduced area, power, and delay time.

Courses in "Mead-Conway" were being offered privately and publicly in an attempt to meet the manpower requirements of the semiconductor electronics industry, which was growing at an exponential rate. Companies were increasing the complexity and volumes of IC's, requiring more and more experienced designers who possessed computer and IC technology knowledge. Traditionally, IC design knowledge was acquired by working under an experienced IC designer in industry. As the need for designers grew, and more powerful CAD tools became available, this mentor style of education was no longer valid. Many North American universities rapidly adopted the "Mead-Conway Design Approach" as a basis for IC design education. Students could now study IC design as part of their undergraduate and graduate education.

In order to send design information to silicon foundries for fabrication, an interchange language called the Caltech Intermediate Form (CIF) was developed by Mead and Conway as a means of describing IC's. CIF is a simple text language which provides for the description of an IC in terms of rectangles, polygons, wires, and round flashes on the various process layers. Symbols are also supported, which can contain combinations of the previously mentioned graphic objects and instantiations (calls) of other symbols. This symbol capability provides a means of describing a structured design in terms of a hierarchy of previously defined subsystems.

In order to educate students in the area of VLSI design using the Mead and Conway methodology, suitable design tools must be available. In the most basic form, students can describe their layouts in terms of CIF code, but this approach defocuses the intent from the design of IC's. When designing an IC, one thinks and works in a three dimensional world of polygons and processing layers. This spatial information is lost when working at a rudimentary text specification level like CIF. Also, when working at the CIF code level, the actual layout process takes much longer as the designer has to iterate between plotting checkplots of the CIF code and using the text editor to make corrections to the CIF code to correct the checkplot. Obviously, working solely in a graphics environment speeds up the

design process.

Industrial quality CAD workstations for IC design cost approximately \$100,000 U.S. If a similar system were to be propagated in a university, then a realistic figure might be \$5,000 per workstation. This criterion made the choice of hardware and software philosophy critical in that one is trying to achieve viable performance with limited resources. By viable, it is meant that the resulting environment must relate to the industrial equivalent (i.e. not a toy). This thesis has achieved this difficult goal.

1.2 Background

When this work commenced, the Department had available one Tektronix 4014 graphics terminal connected to the university's timesharing oriented mainframe computer. A crude program (GENER8) was available for IC layout, but it lacked several features that are required if it was to be used to produce IC's of any complexity. In order to establish an improved design environment, the following options were identified:

1. Extend the GENER8 program and purchase a suitable number of similar Tektronix terminals.
2. Convert an existing program (CIFSYM[7]) to run on the university's mainframe.
3. Develop an application program to run on a stand alone computer.

These three options are discussed below:

Option 1: (Modify an existing graphics based program)

The first option was discarded for two reasons. GENER8 was not designed to be used with the so called "Mead and Conway" design methodology previously described. Converting the program to use CIF code would require about as much effort as starting from scratch. Also, Tektronix 40XX series terminals are prohibitively expensive by today's standards. Spending over \$8,000 for a graphics terminal, which is limited in features to begin with, seemed excessive. Besides, the resulting design environment was not very relevant to the industrial equivalent.

Option 2: (Port an existing ASCII terminal based layout program, eg; CIFSYM)

CIFSYM[7] was a program converted to run on the university's mainframe, using conventional video terminals, (24 rows by 80 columns, and direct cursor addressing), as

graphics terminals (characters corresponded to video pixels). The layout space was traversed by using either a numeric keypad, or the top row of the keyboard, to specify directional movement within the layout space. Program commands were abbreviated to single letters. Though CIFSVM emitted CIF code as the layout definition, it had numerous deficiencies if it was to be used in a mainframe environment. Since all input from the terminal, (including the cursor movement commands), was performed using single character system calls, the program was both slow, in terms of response time, and expensive, in terms of machine resources. Thus, unless the mainframe was lightly loaded, the program was virtually unuseable, particularly if integrated circuit layout was to become a component of undergraduate courses. Student terminals are located throughout the campus in terminal clusters. As machine utilization peaks whenever a first or second year course assignment is nearing the due date, poor system response and terminal access would result during some periods of the week/month. Also, though this program outputs CIF code, it only produces CIF symbols, requiring the user to glue the finished design together using a line oriented text editor. As mentioned above, this was not a desirable method for integrated circuit design, as the designer had to be involved with both three dimensional graphical and the textual definition of the same object. Also, since a CIF symbol is defined in terms of a local coordinate system, and a reference to a CIF symbol instantiates it relative to the surrounding coordinate system, problems arise when working at the CIF code level as to where the actual lower left hand coordinate of the CIF symbol should be placed to achieve the desired result.

For the above reasons, and because the design environment would not closely parallel the industrial equivalent, this option was not pursued further.

Option 3 - (Use a stand alone workstation)

The third option, utilizing a stand alone computer and tools yet to be acquired and developed, was investigated and formed the basis of the three year on-going development which resulted in this thesis. The centralizing capability of the university mainframe has advantages, but the microprocessor approach was seen to be the better choice because of improved interactivity. It should be recognized that the mainframe will always play an important role in VLSI design. Specifically, large scale and numerically intensive simulations belong in that environment as mainframes are on the order of 30 times faster than microprocessors. In this context, for example, a VAX-11/780 would be considered a small mainframe. If a simulation run takes 600 seconds of CPU time on a mainframe, then the

user can expect it to take 18,000 seconds, or 5 hours on a microprocessor. Obviously, this would be unacceptable, and furthermore, unproductive. In this case, the user would be encouraged to transmit the simulation specification files to the mainframe for the run.

Aside from the access problem, using the mainframe alleviates the management and administrative roles that would have to be assumed by the Department. Specifically, the university employs a great many people to staff their computer center, attending to account management, system development and maintenance, equipment acquisition and maintenance for general users, and file storage control including file system backup. Therefore, a design environment which will meet the goals of the thesis will offer a central file storage mechanism for easy file updates and backup along with sufficient processing power to accord users with a fast response time which is crucial for graphics intensive interactive programs. Given recent technology, this goal can now be met by using standalone workstations with a central file serving capability, linked by a local area network. After reviewing the cost of a standalone computer with a graphics capability, it was realized that the cost was within the same order of magnitude as the cost of a graphics terminal which could be connected to the mainframe. The standalone computer has local processing power as an added benefit. With this information in mind, feature requirements for the standalone computer were established.

1.3 Thesis Format

This thesis documents the development of a tutorial environment suitable for the use in teaching CAD based large-scale integrated circuit design. Chapter 2 discusses the hardware requirements for the design environment, Chapter 3 describes the software developed, Chapter 4 presents the results of using the design environment for education, and Chapter 5 presents conclusions and recommendations for future work.

As this thesis represents the development of a tutorial environment for CAD based VLSI education, the appendices contain complete software listings of all the development work described in this thesis. By doing so, it is hoped that this work might form the starting point for other universities with similar intentions in developing CAD based design environments.

Chapter 2 - Hardware Requirements for the VLSI Design Environment

2.0 Introduction

A review of the available electronics engineering periodicals, (Design & Test, Digital Design, VLSI Design ...), revealed that the so called electrical engineering CAD workstations had common features: a Motorola 68000 microprocessor running from 6MHz to 10 MHz, medium to high resolution (780 - 1240 pixels) graphics (usually colour), mass storage, CAD design tools, and cost over \$100,000 U.S. The high cost effectively limits the use of such a workstation as a teaching environment. One could hardly expect to integrate VLSI design into an undergraduate curriculum with one machine and 85 students.

At the graduate level, one machine like that just described, the Metheus workstation, has successfully been used at Carleton in a design course composed of 4 students for one term. There were additional students who were effectively forced away from the machine, and who used the design environment described here.

The following sections identify important areas for consideration when selecting the target computer upon which the design environment was to be built. The target machine is identified, and a comparison to a contemporary machine is made.

2.1 Processor Choice

The processor to be selected cannot be limited to the 64K byte address space, typical of all 8 bit microprocessors. Keeping this in mind, there are two obvious choices - the Motorola 68000 and the Intel 8086 microprocessor families.

2.1.1 The Intel 8086

The Intel 8086 family is a 16 bit microprocessor with an accumulator oriented architecture, and a 1M byte segmented memory space. An offset from a segment register forms the 20 bit address. This strategy poses a serious problem to high level language compilers. Specifically, an array cannot contain more than 64K bytes (which corresponds to a four byte integer array of dimension 128 x 128 elements). A code segment cannot contain more than 64K bytes of code for program counter-relative instructions, as code resides in a code segment which is 64K bytes long. Of course, this is not to suggest that a program is limited to 64K bytes. In fact, the 1M byte address space can be considered to be 16 - 64K byte segments. Special instructions exist to transfer execution from one segment to another.

High level language constructs such as Pascal pointers can be handled if the pointer variable intrinsically carries both the segment register and offset register values. Again, a pointer referenced variable cannot contain more than 64K bytes of data due to the above mentioned restrictions. At the time that the high level language compilers were reviewed, none existed that could address a statically dimensioned array of over 64K bytes, something that a code generator suitably designed could easily handle at the expense of execution speed. Such a code generator could accomplish this by implementing software 32 bit data types with appropriate instructions to manipulate them. The 8086 processor does not have 32 bit instructions, but they can be simulated by using the carry augmented instructions. For example, if one wished to access element 80001 of the byte array called EXAMPLE, then the following scheme could be used. The array index could be divided into the high order 16 bits (\$0001) and the lower order 16 bits (\$3880), to form the 32 bit software data type. The appropriate halves of the base address, (the address of the first element of the array), would be inserted into the data segment and source index registers respectively. If the lower 16 bits of the array index, when added to the source index register, caused an overflow, then the data segment register would have to be modified by the overflow amount. Then, the high order 16 bits of the array index would be shifted left 12 bits before being added to the data segment register. Now, the data segment and source index registers, when added together to form the 20 bit address, contain the physical address of element 80001 of array EXAMPLE. As the reader will note, this is a rather complicated scheme which is not fast in terms of execution speed, as software implemented data types and instructions always execute slower than their hardware counterparts.

2.1.2 The Motorola 68000

The Motorola 68000 family of 16 bit microprocessors is a general purpose register architecture having 9 address, 8 data, and 1 program counter registers, all of which are 32 bits wide. Unlike the 8086, the 68000's 16 megabyte address space is contiguous, and the processor provides for both program counter relative, and absolute address references. Even though the 68000 has a 16 bit data path, it supports 32 bit instructions and operands.

Evaluating high level language compilers for the 68000 revealed that most had rudimentary linkers. These programs restricted code segment sizes to 32K bytes in order to make use of program counter (PC) relative addressing, (which are signed 16 bit displacements), resulting in position independent code reference rather than absolute

addresses which require a more sophisticated optimizing linker, one which was not available. Of course, this restriction results in smaller code, as a relative branch can have either an 8 bit or a 16 bit signed displacement rather than the absolute memory references, requiring a 32 bit address. Specifically, a PC relative branch occupies 2 or 4 bytes corresponding to an 8 bit or 16 bit immediate displacement respectively. Alternatively, an absolute jump requires 1 byte if the destination address is in a register, (which is unlikely), or 5 bytes if the target address is an immediate operand. These addressing restrictions do not apply to data references, providing for very large arrays.

There are more powerful microprocessors which have become available (eg. the 68010, 68020, National 32032) but these chips are to present only found in expensive workstations.

It will be seen later that the 68000 was the only suitable 16 bit processor candidate due to the data addressing capabilities. Basically, the programs we are dealing with here are too large to be convenient on the 8086 architecture.

2.2 Distributed versus Centralized Computing

Once it was clear that a microprocessor approach was to be adopted, the question of single user versus multi user machines (using the microprocessor) was yet to be addressed. Several of the more expensive 68000 based machines, (Chromemco, Dual, Wicat and Radio Shack), offered multiuser capabilities, though these machines could be used as rather expensive single user machines.

If this project was to be successful, the requirements for interactivity had to eliminate one of the most irritating qualities of multiuser timesharing systems - that of unpredictable response time to the user. A single user machine could be slower than a mainframe, depending on loading of the mainframe, but the response time is guaranteed. In other words, if a simulation required 600 seconds on a microprocessor, then it would always require 600 seconds, allowing the user to do something else while waiting. This guaranteed response time requirement effectively eliminates the possibility of selecting a multiuser machine.

In order for software on single user machines to be readily available to all users, a common file system is required, rather than equipping each machine with nonvolatile storage. From an administrative and development perspective, this is desirable as only one copy of an application program exists, rather than copies for each machine, with an obvious advantage

in disk storage savings.

To share a file system amongst many users requires a network to connect all users and file systems together. This leads to adopting a networked microprocessor. The choice adopted made the Department of Electronics among the first to do so in a low cost teaching environment. The two popular network technologies are RS422 (a twisted pair base band scheme operating at 1M Bit per sec.) and Ethernet (a coax base band system operating at either 3 or 10 MBPS). Despite the obvious speed advantage that is offered by Ethernet, its cost is high, and it is not available as an option for all machines. RS422 is inexpensive in that it can use quad telephone cable as its network medium, and the network interface is quite often built into machines which are useable in a network (Corvus, Quark, DY4).

At the time of the beginning of this work, the Corvus Concept was clearly the most appropriate choice given the previous discussion. Since that time, more contemporary machines have become available, as will be discussed in section 2.5.

Operating in a network environment provides the advantage of amortizing costly peripherals (plotters, printers, disk drives) over all user nodes. The alternatives are to equip every node with the peripherals, or to establish a complicated switching system - a "local area network for peripherals". The disadvantage of a networked system is that the hard disk presents a critical point failure for the system.

2.3 Software Support for Program Development

Besides providing file manipulation and operating systems software, if these machines are to be used in a engineering CAD environment, a graphics kernel would have to be available in order to convert/develop an IC layout program. This allows high level language program implementation using standard graphics calls rather than having the application program directly control the graphics hardware. Such kernels usually support more than one graphics device. They manage all the world-to-viewport-to-device coordinate space transformations. Below the graphics kernel lies the device driver, which is responsible for formatting the data in a way which is compatible with the currently selected graphics device.

If one expected the simulation results to agree with those on other machines, floating point calculations software would have to be performed according to the IEEE floating point standard.

Thus, to be able to achieve the goals of this thesis - to produce a CAD environment - the machine should:

1. have good language development tools
2. support the IEEE floating point standard
3. provide medium resolution, (at least 512 by 512 pixel resolution), graphics
4. provide a high level graphics environment for program development

The first three features were available when the machines were purchased. The fourth feature was not, particularly at the time of this work. Thus, a significant portion of this thesis work was the development of a two-dimensional device independent graphics environment.

2.4 The Corvus Concept

The Corvus Concept is a 68000 based standalone computer running at 8MHz, which has 512K bytes of DRAM, an integral monochrome monitor used for both text and graphics, a graphics resolution of 720 by 560 pixels, a built-in RS-422 network interface, two RS232 serial interfaces, and 4 internal expansion slots. The system is unique in that the 14 inch diagonal video monitor may be oriented in either a landscape or portrait mode, and offers approximately 72 pixels per inch resolution in both coordinate directions.

Software support for the Concept includes Fortran and Pascal native code compilers, a 68000 assembler and an object code linker and librarian. The user interface to the operating system is interesting in that it is via soft function keys, (of which there are forty), with dynamically alterable function key annotation on the video display unit, which lends itself to the implementation of hierarchical command structures easily. This is particularly useful for neophyte users in that if the command is not displayed on the label keys, it doesn't exist. This makes for easy use by beginners without documentation, as a function key is the root of a command hierarchy. A more modern equivalent to this is the Apple Macintosh - where a mouse is used to point to menus which contain all the available functions.

The non multi-tasking operating system of the Concept supports many logical windows on the display unit. The user can create up to twenty windows which can encompass any portion of the user area of the video display. All output is sent to the currently selected window. The vendor has provided a library of software to manipulate the

hardware. This library supports the user interfaces including windows, label key annotation, device drivers, along with both local and network connected mass storage devices. The disk servers, (network mass storage devices for file storage), support the notion of pipes as a method of supporting network printers, along with semaphores for ensuring exclusive write access to files by multiple writers. This semaphore feature isn't actually employed by the file management software.

One of the major reasons for selecting this machine was its low cost. At the time, it was the only networked machine available which used a 68000 processor, offered graphics support, and cost less than \$5,000.

2.5 Currently Available Machines

At the time of this thesis writing, more machines have become available which might be considered as candidates for a CAD environment development. One such machine, the Apple Macintosh, is Motorola 68000 based and offers medium resolution graphics and a mouse pointing device.

The Macintosh has been in the market place for over a year now, and costs approximately \$3000 for a 512K byte equipped model. The machine has a 9 inch diagonal video monitor offering monochrome graphics with a resolution of 512 by 342 pixels, corresponding to approximately 80 pixels per inch in both coordinate directions (the Concept is approximately 72 pixels per inch). The processor runs at approximately 6.8 MHz, has much of the low level system support software in ROM, has an integral 3.5 inch disk drive, and has two serial ports, one of which may be used as a network interface to shared peripherals and file storage devices. Network environments are only becoming available at the time of writing of the thesis.

One of the major similarities between the Concept and the Macintosh which make the latter suitable for the design environment is the user interface. As previously mentioned, the Concept offers dynamically alterable function keys, permitting applications programmers to implement command hierarchies by appropriately programming these keys each time the user selects an option. The Macintosh is similar in this regard in that it offers a patented pull down menu feature. The user selects the appropriate menu with the mouse pointing device. A menu item is selected by moving the cursor down the menu selections until the desired option is high-lighted, at which point the mouse button is released, activating the selection. Menu or function key based software is, from our experience, very easy to learn and use,

rarely requiring more than the most rudimentary documentation.

If a decision was to be made today as to which machine was to be selected, one might favor the Macintosh for a number of reasons. The Concept is essentially an original equipment manufacturer's (OEM) machine, to be used by software developers for specialized fields. The machine's manufacturer was responsible for the design and development of the hardware, system software, and applications software. Because of its relative expense, the machine was adapted by few developers, which precluded third party software developers from considering it as a mass market machine to port their software products to. Due to the small dealer base, and the relatively small number of machines purchased, servicing is less than convenient, requiring all of Canada to send machines to be repaired to a firm in North Bay. The Macintosh is a complete contrast to this. Its large installed dealer base results in fast repair or replacement of faulty machines, usually while you wait. Apple provided an operating system which included a software library called the "Toolbox", giving software developers access to the graphics, peripherals (including the mouse), and file system at a very high level. For instance, the user does not concern himself whether or not the mouse button is currently down, but whether the event handler has detected a mouse event. Apple wrote a few basic applications, (MacWrite, MacPaint, MacTerminal, etc), but heavily relied on third party software developers to make the machine a success, by developing new applications or porting popular existing applications from other machines. We are now seeing electronics oriented applications running on the Macintosh. To date, other than the applications which have been ported or developed for the Concept as a result of this thesis, nothing is available which could be used in the instruction of CAD based VLSI design on the Concept. Although it is outside the scope of this thesis, it would seem appropriate to transfer the design environment developed here to the Macintosh or other machines offering similar user interfaces.

Chapter 3 - Software Requirements for the VLSI Design Environment

3.0 Introduction

As mentioned in Chapter 1, to be able to design an integrated circuit effectively requires computer based design tools. At a minimum, the requirements are an interactive graphics editor to do the IC layout, and a circuit simulator to establish that components of a design are correct according to the design criteria and the simulation results.

The following sections describe

- (i) the design and implementation of a two-dimensional device independent graphics library.
- (ii) the design and implementation of an interactive IC layout editor (CifDraw) using the above graphics library.
- (iii) the porting and conversion of the defacto standard device-level circuit simulation program SPICE, available from the University of California at Berkeley. This program is necessary for detailed analog simulation in the time and frequency domains, as well as precise timing simulation for digital circuits at the device level.
- (iv) the porting and conversion of a public domain logic and mixed-mode simulator, SPLICE, also available from the University of California at Berkeley.
- (v) the design and implementation of an continuous frequency domain filter response program used for tutorial puposes.

these devices regardless of their resolution differences (for example, the LA120 has 110 pixels per inch in the horizontal direction and 92 pixels per inch in the vertical direction, while the Concept has 72 pixels per inch in both directions). Descriptions of the individual device drivers follows in a later section.

String255 is the type declaration for passing character strings to the text routines.

3.1.3 Interface Definitions

The following section documents the interface routines that an application programmer would use to produce graphic output.

Function InitGraphics (Device: DeviceType): Boolean;

This function, when called with the appropriate device name as described above, will attempt to establish a connection between the graphics system and the device to be selected. The function will return a true result if this attempt is successful. If the HP7221 is selected, the device driver will automatically attempt to load and suitably initialize the appropriate RS-232C port on the Concept base unit (DtaCom1). If the LA120 device is selected, the device driver will open a file called PLOT.TEXT into which the rasterized output will be retained.

Procedure TermGraphics;

This procedure terminates the currently open device stream. In the case of the HP7221, any pending buffers will be dispatched to the device, while in the case of the LA120, the vector to raster scan conversion will be completed, and the output file closed. Once this routine has executed, further graphics calls will fail until a device is attached using the InitGraphics routine. If the application program is overlayed, the user might wish to call this routine as soon as the graphic generation phase is complete, which will allow the device driver to be unloaded from memory. The device drivers reside in their own segment and remain in memory from the time InitGraphics is called until TermGraphics is called.

Procedure InqDevice (Var XMin, XMax, YMin, YMax, XDotsPerInch, YDotsPerInch: Integer);

Returns the current physical device coordinate space dimensions along with the number of pixels per inch in the horizontal and vertical directions.

the device resolution is the same in both the horizontal and vertical directions, or how information is to be formatted to obtain the desired output. Also, a system of this nature guarantees that the same image, (in the correct perspective), may be obtained from all devices for which device drivers exist. Thus, the user need only select the appropriate device, and the package will then scale and transform the data correctly. Future device support is relatively simple, only a device driver, (the code which takes the low level primitive data and produces the corresponding output), is required - the rest of the graphics package is static.

As the diagram above indicates, the first transformation is between the world and viewport coordinate spaces. As can be seen, the viewport can be the whole virtual device space, or some fraction of it. This has uses when a composite graphic is to be formed from various world coordinate space images.

The virtual device coordinate space, (sometimes referred to as the normalized device coordinate space or NDC space), is then mapped onto the currently selected physical device, which of course, has its own coordinate space.

The package currently supports one active device at one time, thus these coordinate transformations can be concatenated into one composite transformation, mapping the world coordinates directly onto the device, once the various transformation and scaling factors have been defined. Multiple active logical and physical devices could easily be supported by identifying each device by a stream identifier, which would identify the appropriate transformation matrix to apply to the world coordinate data.

The graphics package is also responsible for ensuring that only valid data reaches the currently active device driver. This is accomplished by clipping world coordinate data against the coordinate space defined by the user, ensuring that only visible data reaches the driver. In the interests of execution speed, the graphics package makes extensive use of 16 bit integer arithmetic in order to take advantage of the hardware multiply and divide instructions.

3.1.2 User Interface

The user is currently provided with twenty-one Pascal[4] interface routines to the graphics package. These are individually described below and include routines for opening and closing a stream to a device, absolute and relative point, line, and text routines using the currently selected line type and colour, and software and hardware text facilities, where the former is used in cases where the device doesn't support graphical text output. The

following diagram illustrates how the graphics library is used:

Figure 3 - Graphics System Routine Organization

As the diagram suggests, the graphics library entry points may be divided into three categories, which will be individually discussed below.

The first category - Graphics System Initialization - contains the routines required to establish the graphics environment. The user first selects which graphic output device is to be used by calling the InitGraphics routine. Information specific to the selected device, (dimensions and resolution of device in device units), may be solicited by using the InqDevice routine. The user may optionally specify a portion of the device to be used for the graphic generation, rather than the whole device drawing area, by calling the SetDevice routine. The SetViewport routine is used to specify a viewport into the normalized virtual device coordinate space as illustrated in figure 2. The user may change the default world coordinate space dimensions by calling the SetWindow procedure.

The second category - Graphic Generation - contains procedures to draw lines, points, and text on the currently selected output device. For each of these object types, both an absolute and relative movement routine version exists. The relative movements are all specified relative to the last SlewAbs procedure call. Routines also exist to enquire and set the current line style and colour.

The last category - Graphics System Termination - contains one routine which is responsible for closing the currently selected graphics device. In the case of the HP plotter,

any pending output is dispatched to the device. If the LA120 DecWriter is selected, then the vector-to-raster scan conversion process, (which is described in detail in section 3.1.5.1), can commence. Section 3.1.3 describes the user interface routines in detail.

3.1.2.1 Type Declarations

The version of Pascal used for this development work permits separately compiled units. The following information is contained in a unit called GraphDefs in the library called /ccutil/graphlib. The user is expected to include the following information in the top of the applications program:

```
Program Whatever (Input, Output);
Uses
  {$U /ccutil/GraphLib}
  GraphDefs,
  Graph;
```

Doing so will give the applications programmer access to the following type declarations:

Type		
ScreenColor	=	(None, Black, Brown, Blue, Green, Lime, Red, Orange, Yellow, White, Flip);
LineType	=	(Dotted, Dashed, Solid);
DeviceType	=	(Concept, HP7221, LA120, NoDevice);
String255	=	String [255];

ScreenColour enumerates all the line colours currently supported. These have meaning only for the HP7221 eight pen flat bed plotter. The LA120 DecWriter equipped with the Selanar graphics board supports only black and white lines, while the Concept video screen supports black, white and flip colours, where flip toggles a pixel from white to black or black to white depending on the pixels previous state.

Currently, three line types are supported for the HP plotter, while the Concept supports dotted lines in the horizontal and vertical directions only. All devices support solid lines.

Three devices are supported - an HP7221 plotter, an LA120 lineprinter equipped with a Selanar graphics board, and the Concept video screen. As previously mentioned, a graphic (an image created in terms of point, line, and text primitives) can be displayed on any of

these devices regardless of their resolution differences (for example, the LA120 has 110 pixels per inch in the horizontal direction and 110 pixels per inch in the vertical direction, while the Concept has 72 pixels per inch in both directions). Descriptions of the individual device drivers follows in a later section.

String255 is the type declaration for passing character strings to the text routines.

3.1.3 Interface Definitions

The following section documents the interface routines that an application programmer would use to produce graphic output.

Function InitGraphics (Device: DeviceType): Boolean;

This function, when called with the appropriate device name as described above, will attempt to establish a connection between the graphics system and the device to be selected. The function will return a true result if this attempt is successful. If the HP7221 is selected, the device driver will automatically attempt to load and suitably initialize the appropriate RS-232C port on the Concept base unit (DtaCom1). If the LA120 device is selected, the device driver will open a file called PLOT.TEXT into which the rasterized output will be retained.

Procedure TermGraphics;

This procedure terminates the currently open device stream. In the case of the HP7221, any pending buffers will be dispatched to the device, while in the case of the LA120, the vector to raster scan conversion will be completed, and the output file closed. Once this routine has executed, further graphics calls will fail until a device is attached using the InitGraphics routine. If the application program is overlayed, the user might wish to call this routine as soon as the graphic generation phase is complete, which will allow the device driver to be unloaded from memory. The device drivers reside in their own segment and remain in memory from the time InitGraphics is called until TermGraphics is called.

Procedure InqDevice (Var XMin, XMax, YMin, YMax, XDotsPerInch, YDotsPerInch: Integer);

Returns the current physical device coordinate space dimensions along with the number of pixels per inch in the horizontal and vertical directions.

Procedure SetWindow (XMin, XMax, YMin, YMax: Integer):

Sets the world coordinate space dimensions according to the values of the minimum and maximum X and Y values. In this implementation, integer variables occupy sixteen bits. X and Y may range between -32768 to +32767.

Procedure SetViewport (XMin, XMax, YMin, YMax: Real):

Sets the fraction of the virtual device coordinate space to be associated with the current viewport. X and Y minimum and maximum may take on values in the range 0.0 to 1.0

Procedure SetDevice (Var XMin, XMax, YMin, YMax: Integer):

Sets the portion of the physical device coordinate space to be used for graph output. Notice that the procedure expects var type parameters as the procedure returns the actual values used in the event that invalid parameters are submitted.

Procedure NewPage:

This routine will erase the screen if the Concept screen is the current output device, dispatch any pending buffers if the HP7221 plotter is being used, or issue a linefeed if the LA120 line printer is being used.

Procedure SetColour (Colour: ScreenColour):

Sets the current line colour. All lines generated after this call will assume this new value.

Procedure InqColour (Var Colour: ScreenColour):

Returns the current line colour.

Procedure SetLine (Style: LineType):

Sets the current line style. All lines generated after this call will assume the selected style assuming that the device can support them according to the restrictions mentioned under the type declarations above.

Procedure InqLine (Var Style: LineType):

Returns the current line style.

Procedure SlewAbs (X, Y: Integer):

This procedure lifts and moves the writing beam to the specified coordinate. The old X and Y absolute values are updated to reflect these new values, and the relative X and Y values are set to zero

Procedure SlewRel (X, Y: Integer):

This procedure lifts and moves the writing beam to the specified coordinate. The old X and Y absolute values are not updated, but the relative X and Y values are updated to reflect these new values.

Procedure DrawAbs (X, Y: Integer):

This procedure draws a line of the specified colour and line style between the previously specified coordinate using the SlewAbs procedure and the coordinate specified by the parameters to this procedure. Then the old X and Y values are updated to reflect these new values. For example, to draw a rectangle between {0,0} and {10,10}, one would SlewAbs(0,0) to move the beam to the lower left hand corner of the rectangle. This would be followed by DrawAbs(10,0), DrawAbs(10,10), DrawAbs(0,10), and finally, DrawAbs(0,0). It should be noted that the relative X and Y values are set to zero.

Procedure DrawRel (X, Y: Integer):

This procedure draws a line from the previously established relative X and Y values and the new values. Then the relative X and Y values are set to these new values. The old X and Y values are not modified. This routine is useful, for example, to draw an object of a constant size anywhere in the world coordinate space. For example, one could again draw a rectangle with width and height of ten world coordinate units, having a lower left hand corner at the coordinate {100,200} by using the following specifications: SlewAbs(100,200), DrawRel(10,0), DrawRel(10,10), DrawRel(0,10), DrawRel(0,0).

Procedure PointAbs (X, Y: Integer):

This routine plots a point using the current colour, at the specified coordinate. The old values of X and Y are set to this new value. The relative X and Y values are set to zero.

Procedure PointRel (X, Y: Integer):

This routine draws a point relative to the previously established X and Y values. For example, one could plot a point at {12,16} relative to the coordinate {345,632} using the following specifications: SlewAbs(345,632) followed by PointRel(12,16).

Procedure LineAbs (X_1, Y_1, X_2, Y_2: Integer):

This routine draws a line using the current line style and colour between the world coordinates {X_1,Y_1} and {X_2,Y_2}. The old values of X and Y are set to {X_2,Y_2}, and the relative values of X and Y are set to zero.

Procedure LineRel (X_1, Y_1, X_2, Y_2: Integer):

This routine draws a line relative to the previously established realative X and Y values, using the current line style and colour. The relative X and Y values are then set to {X_2,Y_2}. For example, to draw a line from {20,35} to {40,45} relative to the point {700,785}, one would use the following specification: SlewAbs(700,785) followed by LineRel(20,35,40,45).

Procedure TextAbs (X, Y: Integer; Chars: String255; Theta, Slant, Width, Height: Integer):

This routine will plot the text string (Chars), placing the lower left hand corner of the first character at the point {X,Y}, at an angle relative to the horizontal (Theta) and a slant angle (Slant). The text will have a character width and height of Width and Height world coordinate space units respectively. Once the text string has been plotted, th absolute X and Y values will be updated to coincide with the coordinate of the lower right hand corner of the end of the text string. The relative X and Y values will be set to zero. For the Concept and the LA120, software characters are used, while hardware characters are used for the HP7221 plotter.

Procedure TextRel (X, Y: Integer; Chars: String255; Theta, Slant, Width, Height: Integer):

This routine will plot the text string (Chars), placing the lower left hand corner of the first character at location {X,Y} relative to the previously established absolute X and Y values, at an angle relative to the horizontal (Theta) and a slant angle (Slant). The text will have a character width and height of Width and Height world coordinate space units

respectively. The relative X and Y values are not modified. Again, for the Concept and the LA120, software characters are used, while the HP7221 utilizes its hardware resident fonts.

Appendices A-1 to A-5 contain complete software listings of the graphics system and graphics definitions files.

3.1.4 Implementation Details

To implement a graphics system such as this required that a device independent interface be provided to the user, while allowing continued updates and maintenance to the graphics system. To achieve such a system, a "front end" was created which is responsible for all data transformation down to the device driver interface. Doing so permits additional device drivers to be added without changing the applications program interface to the graphics system. If previously created programs are to utilize newly offered device support, the applications program must only recompile and relink the application program to use the newly offered device(s).

3.1.4.1 Device Driver Interface Specifications

Currently, a device driver is not run time loaded in the strict sense of the word. Rather, every routine which talks to a device driver directly, of which there are eight, has a case statement, which uses the currently selected device as the case selection expression. There must be an appropriate driver interface call to support each device referenced in the case statement. The following describes the current interface definition. By convention, DEVICE is replaced by some suitable string describing the device being interfaced. Currently DEVICE is one of Concept, HP7221, and LA120. It should be mentioned that all coordinate references are in terms of the device, not the world coordinate space. By the time data gets to the device driver, it has already been transformed by the graphics system front end.

Function Init_DEVICE (Colour: ScreenColour, Style: LineType): Boolean;

• The graphics system calls this device interface routine to initialize the driver for future graphic output. The driver uses this interface to initialize private data structures, initialize hardware specific to the interface, and possibly lock the device driver into memory. The

driver should return true if the initialization was successful. Colour and Style are the values for all graphics created until the application program issues an SetColour and/or SetStyle directive.

Procedure Inq_DEVICE (Var XMin, XMax, YMin, YMax, XDotsPerInch,
YDotsPerInch: Integer);

The graphics system calls this driver interface routine to solicit information to initialize the virtual device to physical device transformation matrix. The device driver is expected to return the current values of the lower left and upper right hand corner of the physical device coordinate space along with the device resolution in pizels per inch except for the HP7221, which returns values in millimeters. The Concept device driver returns information for the currently selected window, which is usually the whole user window. This has the convenient side effect that the information is valid regardless of the video display monitor (it can be oriented in the portrait or landscape mode).

Procedure Set_DEVICE (Var XMin, XMax, YMin, YMax: Integer);

The graphics system calls this routine to specify what portion of the device is to be used for future output. Note that the parameters are passed by address, allowing the routine to return the actual values used in cases of incorrectly specified values.

Procedure Colour_DEVICE (Colour: ScreenColour);

The graphics system calls this interface routine to indicate the current line colour to be used for future graphic generation.

Procedure Style_DEVICE (Style: LineType);

The graphics sytem calls this interface routine to indicate the current line style to be used for future graphic generation.

Procedure Move_DEVICE (X, Y: Integer);

This routine is used to indicate the initial coordinate for a device draw command. A local pair of variables called X_Old and Y_old are used to retain this value, used by the HP7221 driver to optimize connected line graphics of the same line style and colour without raising the pen off of the paper.

Procedure Draw_DEVICE (X, Y: Integer):

This routine draws a line from {X_Old, Y_Old} to {X,Y} using the current value of line type and colour, and then sets {X_Old,Y_Old} to {X,Y}.

Procedure Point_DEVICE (X, Y: Integer):

This routine draws a point as {X,Y}, and then sets {X_Old,Y_Old} to {X,Y}. The current value of colour is used if the driver can support this feature.

Procedure Line_DEVICE (X_1, Y_1, X_2, Y_2: Integer):

This routine is called to draw a line using the current value of colour and line style, between {X_1,Y_1} and {X_2,Y_2}. {X_Old,Y_Old} are then set to {X,Y}.

Procedure Text_DEVICE (X, Y: Integer; Chars: String255; Theta, Slant: Real; Width, Height: Integer):

This is called to utilize hardware text, if the device supports it. Otherwise, a stub must be provided for this routine. {X,Y} is the lower left hand corner of the first character of the character string (Chars). The real variables Theta and Slant refer to the angle of rotation of the character string from the horizontal, and the slant angle of the characters respectively. Width and Height specify the character size in device coordinates.

Procedure Page_DEVICE:

This routine is called to issue a page command to the device. For video type displays, the screen is to be cleared, while a page eject is to be performed on hard copy devices. In the case where a hardware device cannot perform a page eject, (for example - HP7221 plotter), then any pending output queued should be dispatched to the device.

Procedure Term_DEVICE:

This is called to close the channel to the device driver, possibly allowing the device driver to be unloaded from memory. In the case of raster type hard copy devices, (for example - an LA120 equipped with a Selanar graphics board), a call to this routine would cause the vector to raster scan conversion to occur. The vector to raster scan conversion process can't start until all the graphic material has been submitted as the graphics, by definition, is a random access process.

Appendices A-3 to A-5 contain the device driver software listings for the Concept, LA120, and HP7221 devices.

3.1.4.2 Software Character Definition

As previously mentioned, the graphics library software text is used in cases where the desired device does not support hardware text. The software text routine was designed in a manner which permits easy character definition. Software characters are defined on a seven by seven grid, with a two-pixel whitespace on the right hand side of the character. Creating a new character definition involves entering the character to be associated with the graphic definition in the list of all characters supported, and then defining the graphic equivalent in terms of slews and draws. To automate this procedure, a program called Create was written to define the software characters. Appendix A-6 contains a listing of this program.

3.1.5 Device Driver Implementation Details

There are basically two classes of graphics devices - vector devices and raster devices. A vector device is one in which the end points of a vector can be specified, and the device will draw it. A memory mapped video display and a plotter are two examples of such devices. A raster device is one which must have all elements decomposed into rasters, such that when the current raster line intersects an active element, it is drawn. Once the active raster has passed by, it cannot be backed up to catch any missed data. An example of such a device is a DecWriter (LA120) equipped with a Selanar graphics board. The active raster is taken to be the print head relative to the paper. Once the paper is advanced past the print head, it cannot be backed up. This presents a problem.

The Concept manufacturer provided an interface to the video display driver, which resulted in little work to draw vectors and points on the screen. Likewise, the plotter manufacturer provided data encoding flow charts which were implemented with minor problems. There were no sources available for doing vector to raster scan conversion, and the following describes a novel, high language vector to raster scan conversion program that was designed, developed and implemented.

3.1.5.1 Vector to Raster Scan Conversion

Vector to raster scan conversion can be separated into two distinct phases. The first phase comprises the graphic generation in terms of vectors and points. During this phase,

the vectors are inserted in a sorted list, ordered on their maximum Y coordinates as they are generated. Once the graphic generation is complete, the vector to raster scan conversion phase is completed. This involves commencing at a Y coordinate which is greater than or equal to the largest sorted Y value in the vector list, (points are simply a degenerate case of vectors). For the current Y value, all elements are inspected to see if they intersect this current scan line. If they do so, a pixel is produced in the appropriate X location buffer. Once all the elements for the current scan line have been converted, the scan buffer is dispatched to the device, and the process is repeated for the next smaller value of Y. This continues until the complete graphic has been reduced to rasters.

Vector to raster scan conversion may be accomplished using two methods, depending on the magnitude of the output. For simple graphics, (say 24 by 80 pixels, not unlike that of a video terminal), one may use an appropriately dimensioned array, with each element corresponding to the equivalent screen coordinates. The graphic is created by randomly addressing the array as required. The graphic is then displayed by dispatching the top row of the array first, then the second, and so on, until all the rows have been output. This method is unsatisfactory as the resolution size product of the output device increases. When using a video terminal, it has a resolution of 80 columns by 24 rows, for a total of 1920 pixels, (240 bytes if one cell corresponds to one bit rather than one byte). When using a DecWriter III equipped with Selanar graphics board, the device has a resolution of 110 pixels per inch in the X direction, and 72 pixels per inch in the Y direction, where a pixel is taken to be one bit. Using this device to display a ten inch by ten inch graphic would require 792000 pixels, or 99000 bytes of storage. Clearly, high resolution and size output devices require a less storage hungry approach. A standard solution to a problem of this nature is vector to raster scan conversion on the fly. Lines and points, (the most fundamental components of a graphic), may be encoded and sorted such that they fall below or on the active scan line (the current horizontal line crossing the graphic corresponding to the printhead direction of travel). It is necessary to use such a scheme due to the unidirectional paper movement of the output device. In such a situation, a random access line buffer is maintained, which is used to scan convert all active elements (those that fall on or across the active scan line). Once this has been accomplished, the line buffer is dispatched to the output device, the scan line is decreased by one, and the process is repeated again. This repeats until all the graphic objects have been decomposed into the appropriate rasters. The following diagram illustrates this process:

Figure 4 - Vector To Raster Scan Conversion Process

To implement such a mechanism requires a fast line drawing algorithm which results in "nice" lines. Clearly, points are a trivial case once they have been transformed from the world to device coordinate spaces. Newman and Sproull[5] present a symmetrical digital differential algorithm (DDA), which may be implemented to result in fast line generation.

3.1.5.2 Line Drawing Algorithm

Newman and Sproull[5] present the DDA algorithm, which was developed from the first order difference equations describing the slope of a line. Specifically:

$$\frac{dy}{dx} = \frac{\text{delta_y}}{\text{delta_x}}$$

If the current x and y coordinates of the line being generated are maintained in separate registers, then for each pixel describing the line, the registers are incremented by an amount proportional to the derivatives of x and y, where the integer portion of the register represents the pixel coordinate. In [5], the algorithm is described as follows:

Given that one wishes to draw a line from $\{x_1, y_1\}$ to $\{x_2, y_2\}$, then satisfy the following equation:

$$2^{n-1} \leq \max(\text{Abs}(\text{delta_x}), \text{Abs}(\text{delta_y})) < 2^n \quad (1)$$

where $\text{delta_x} = x_2 - x_1$, and $\text{delta_y} = y_2 - y_1$.

In the case of the symmetrical DDA, define $\epsilon = 2^{-n}$. The line will be generated by incrementing the running X and Y registers by $\epsilon * \Delta_x$ and $\epsilon * \Delta_y$ respectively. Thus, it can be seen that the line will be composed of $(1+2^n)$ not necessarily unique pixels. As mentioned in [5], the actual line drawing maybe done with two additions per pixel generated if the initial values of X and Y are shifted left by n bits. This might be undesirable from a high level language point of view in that the running X and Y values would need to be shifted right by n bits to extract the integer portion of the pixel coordinate. Rather than this, one can make use of the underlying machine architecture by aligning the decimal point on an integral storage unit boundary, (ie. on a byte, word, or long word boundary), by suitably shifting the $\epsilon * \Delta_x$ and $\epsilon * \Delta_y$ values such that the resultant denominator is 2^x , where x is the width in bits of the fraction. This results in a rational fraction. The integer portion of these variables can be accessed by defining the storage unit to one size for the purpose of the addition, and another size for accessing the pixel coordinate. This can be accomplished by using a Pascal[4] variant record as shown below:

```

Type_DDA    = Record
              Case Boolean Of
                True: (Bits32:      LongInt);
                False: (Whole:      Integer,
                        Fract:      Integer);
              End;

```

The following diagram illustrates the memory overlapping described by the above declaration:

Figure 5 - Memory Image Of Overlapping Data Structures

The $\epsilon * \Delta$ values are actually shifted left by:

STORAGE_SIZE_IN_BITS - n

where n was found by (1), and the storage size is the width of the portion of the running X and Y values associated with the fractional component, in bits. The value of n is 16 in the figure above, corresponding to a denominator of 65536. A beneficial side effect of this

scheme is the fact that the $\epsilon \cdot \Delta$ values are stored as two's complement numbers, and thus, subtraction is performed by addition. No logic is required to detect if the number is negative or positive. The reader will note that the addressable range and line length in pixels is obviously a function of the chosen storage class size. Specifically, the maximum line length in pixels is $1+2^n$ (where n is the storage size in bits of the fractional part of the running register). The following pseudo code sketches out the algorithm:

```

delta_x := x2 - x1
delta_y := y2 - y1;
Choose N such that :
    2 ** (N-1) <= max (abs (delta_x), abs (delta_y)) < 2 ** N
epsilon_delta_x := delta_x / 2 ** N;
epsilon_delta_y := delta_y / 2 ** N;
Current_X := x1 + 0.5;
Current_Y := y1 + 0.5;
For I := 0 To 2 ** N
    plot_pixel (int (Current_X), int (Current_Y));
    Current_X := Current_X + epsilon_delta_x;
    Current_Y := Current_Y + epsilon_delta_y;
EndFor

```

As can be seen, if $\epsilon \cdot \Delta_x$ were retained as a thirty-two bit quantity, where only the lower sixteen bits of the variable were used to represent the fraction, (the sign is not part of this space), then the thirty-two bit variable could be incremented, then the appropriate integer part removed. The following illustrates this point:

The following variable declarations are assumed:

```

Var
    Current_X:      Type_DDA;
    Current_Y:      Type_DDA;
    Epsilon_Delta_X: LongInt;
    Epsilon_Delta_Y: LongInt;

```

To increment the registers:

```

Current_X.Bits32 := Current_X.Bits32 + Epsilon_Delta_X;
Current_Y.Bits32 := Current_Y.Bits32 + Epsilon_Delta_Y;

```

To extract the whole parts of the variables in order to plot the pixel:

```

Plot_Pixel (Current_X.Whole, Current_Y.Whole);

```

The following example illustrates the process of drawing a line from {1,5} to {6,3}.

```

n = 3 ->    EdX = +5/8
            EdY = -2/8

```


Initially, X and Y have the following values:

	<u>Integer</u>	<u>Fraction</u>		<u>Integer</u>	<u>Fraction</u>
X:	0001	.1000	Y:	0101	.1000
EdX:	0000	.1010	EdY:	1111	.1100

The line will be formed from nine pixels, (2^3+1) , as shown below. The values in parentheses represent the integer result of the operation, the value used to define the pixel coordinate.

1.	0001.1000 (1)	0101.1000 (5)
±	<u>0000.1010</u>	± <u>1111.1100</u>
2.	0010.0010 (2)	0101.0100 (5)
±	<u>0000.1010</u>	± <u>1111.1100</u>
3.	0010.1100 (2)	0101.0000 (5)
±	<u>0000.1010</u>	± <u>1111.1100</u>
4.	0011.0110 (3)	0100.1100 (4)
±	<u>0000.1010</u>	± <u>1111.1100</u>
5.	0100.0000 (4)	0100.1000 (4)
±	<u>0000.1010</u>	± <u>1111.1100</u>
6.	0100.1010 (4)	0100.0100 (4)
±	<u>0000.1010</u>	± <u>1111.1100</u>
7.	0101.0100 (5)	0100.0000 (4)
±	<u>0000.1010</u>	± <u>1111.1100</u>
8.	0101.1110 (5)	0011.1100 (3)
±	<u>0000.1010</u>	± <u>1111.1100</u>
9.	0110.1000 (6)	0011.1000 (3)

The following diagram illustrates the resultant line generation.

Figure 6 - Resultant Rasterized Image

3.2 IC Layout

3.2.0 Introduction

Once the high level graphics support was reliable the IC layout program could be designed and implemented. The Concept may be considered as being both a computer and a medium resolution graphics terminal. With this in mind, it was decided that an interactive graphics layout program for integrated circuits was to be developed. Specifically, the program was to provide a designer with fast response in terms of graphical drawing speed of reasonably complex designs, once the design had been loaded into the local memory of the Concept, while relying solely on the local processing power of the machine. As the Concepts are configured in a local area network (LAN) sharing a common hard disk, an important goal was to attempt to maintain all code and data memory resident except when either the program or designs were being loaded from the disk server. This would result in little network traffic during the actual design phase, thus not affecting other users on the baseband system. As will be seen later, attempts were also successfully implemented to rapidly access data currently visible on the graphics screen.

3.2.1 CifDraw Description

After some preliminary program initialization, the user is presented with the following video display:

Figure 7 - CifDraw Video Screen

The operating system supported function keys are organized into two groups. The primary label set, (the ones not preceded by Command-), is the default display, and occupies the upper two rows of the diagram above (eg.; Delete and Mark). The secondary set replaces the primary set when the keyboard <Command> key is depressed (eg.; Set Visible). Many of the commands lead to secondary function key label sets, essentially

implementing a hierarchical menu via programmable function keys. The function key annotation labels occupy an area at the bottom of the display not accessible by applications programs. Immediately above the function key annotation area, in the command window, is a status display which reports information which includes: the current dynamic cursor coordinates (the cross hair), the absolute difference between the dynamic and static cursor coordinates, the currently active layer upon which rectangles will be drawn, and the currently visible layers. With the exception of being prompted for file names or layout space coordinates, the user communicates via the function and cursor keys only. This virtually eliminates the requirement to remember commands - if the command to be executed does not exist on one of the function key labels, then the command does not exist. This is a particularly valuable feature in a tutorial environment where it is desirable to have students using the design tool with a minimal amount of instruction.

The program uses the currently selected display window, and operates in both the portrait and landscape modes to suit the user's taste.

The user traverses the layout space, (which is 4800 by 4800 lambda), by manipulating the cursor control keys located on the keyboard, which control the position of the dynamic cursor (the cross hair cursor). Alternatively, the user may directly position the dynamic cursor at any valid coordinate by using the <Goto> command, which prompts for the coordinates to which the cursor is to be moved. When a screen boundary is encountered, and there is still layout space past the boundary, the screen is erased, and redrawn to reposition the display area such that some fraction of the original and new areas are displayed. It was empirically determined that it was faster to erase and redraw the screen rather than scroll the screen in the appropriate direction, rolling in the new area on the leading edge. If a screen boundary is encountered, and there is no remaining area past the boundary in the direction of cursor movement, the terminal bell is rung to indicate the error. The fraction of new and old areas displayed on the video screen as a result of hitting a screen boundary is controlled by the <Set-ScmSlew> command. Initially, the update value is 1/2 which corresponds to half a new screen update in the direction of cursor movement when a screen boundary is encountered.

Based on the Manhattan system for photo-lithography mask definition, all graphical objects are rectangles oriented parallel to the X axis. Using these primitives, a user can create complex polygonal geometries by butting appropriately sized rectangles. To create a rectangle, the user moves the dynamic cursor, (the cross hair), via the keyboard cursor keys,

to one of the desired corners of the rectangle. The <Slew> command then causes the static cursor, (the diamond cursor), to be moved to the same coordinates as those of the dynamic cursor, permitting the user to then position the dynamic cursor at the opposite corner of the desired rectangle. The <Mark> command actually creates the rectangle on the currently selected level, (in the case of the screen layout above, a 40 by 20 lambda rectangle would be placed on the P-Well layer centered about the coordinate {60,50}). Graphical objects are deleted in a similar manner by using the <Delete> command instead of the <Mark> command, again, on the currently selected layer. During deletions, rather than redrawing the screen each time a rectangle is deleted, the deleted object is visually marked as being deleted by drawing a cross through it. This approach was adopted as it was discovered that objects could not easily be removed from the screen without retaining much more graphical information about their locations. The user may explicitly update the screen by using the <ReDraw> command.

The user is provided with two cursor movement speeds. <SlewSlow> causes the dynamic cursor to move at the default, (or user specified by the <Set SlowSlew> command), rate of one lambda per cursor key hit. The faster cursor movement speed may be selected by the <SlewFast> command which has a default speed of ten lambda per cursor key hit. Again, the fast slew speed may be altered by the user, by using the <Set FastSlew> command. The cursor may be directly positioned anywhere in the layout space by using the <Goto> command. The user is prompted for the coordinates, and then the program positions the cursor to the specified location. The video display screen is conveniently updated such that the coordinate moved to falls in the center of the screen.

As previously mentioned, rectangles are instantiated on the currently selected layer, which is selected by using the <Set Current> command. The selected layer is indicated in reverse video over the set of all currently visible layers in the status window area. The user can define which layers are to be visible by using the <Set Visible> command. All layers which are visible are marked in inverse video in the visible layer status line in the status window area. All layers not visible are indicated in normal video. Also, rectangles created on the currently selected layer are drawn using dotted lines, while all others are drawn using solid lines. This scheme was chosen in preference to shading rectangles according to the layer they were instantiated on. An early prototype of the program supported shading, but it executed much too slowly. To have the program execute as fast as possible, while still retaining a transportable design, pre-empts the use of any graphics from being created by

means other than those supported by the transportable graphics library. If this design philosophy was not adhered to, then 68000 machine language routines could be written to directly control the graphics screen.

The minimum cursor movement increment is one lambda which corresponds to one micron for our implementation. This differs from Mead and Conway, where one lambda corresponds to 0.01 micron. Also, because the user specifies the opposite corners of a rectangle, the program must internally maintain a geometry list whose coordinates are twice as fine as user's. This was done to accommodate the CIF definition of a rectangle, which includes the coordinate of the rectangle's center, width, height, and direction vector, which are all specified as integer values. Obvious problems occur if the center point of the rectangle falls on a half CIF unit. Therefore, to scale the CIF symbols to correspond to Mead and Conway's definition, it is necessary for the user to include a scaling factor of 100/2, which maps the internal coordinates to microns.

CifDraw permits the user to create symbols only. To address the desire of hierarchical design, the program permits the user to create a flat file, (i.e., fully instantiated), by reading in previously created symbols. The user may, for example, create a multiple stage shift register by first creating a single stage as one symbol. Then, the user instantiates this symbol by positioning the dynamic cursor to the lower left hand corner of the desired symbol placement. <GetFile> is then used to read in the symbol file, thus creating another copy of the symbol in the current layout space. When the design is finished, the user is left with a monolithic symbol. <SaveFile> is used to save the current layout space context. If the user attempts to leave the program by using the <Exit> command, having not previously saved the layout space information, the program will prompt him for a file to save the CIF symbol in. A null entry for the file name defeats this feature. <ListFile> is provided to list the file contents of any volume to which the user has access to. This is useful for determining if the current layout space has been saved, or if there is enough space in a volume to save the layout space.

The program currently provides hard copy support for the HP7221 pen plotter. When plotting, the program uses the currently visible layers, and layout space magnification. This permits users to plot the metal layer with contacts only, for rudimentary design rule checking. The plotter selects the pen colours according to the Canadian Microelectronics Corporation specifications. When the <Plot> command is initially selected, the user is prompted for a page size, and the lower left and upper right coordinates of a rectangle which

bounds the area to be plotted. The program then plots the layout space, possibly using more than one sheet of paper. Alignment marks are drawn on each sheet of paper to assist the user in creating a multiple page plot. The user may abort the plotting process at any time by using the <Exit> command. Otherwise, once the next sheet of paper has been inserted, the user simply issues the <Continue> command, and the next page of the plot is produced.

CifDraw offers various environmental features. The user may specify the current layout space magnification. Using the default magnification of one, the user is able to view a 136 by 159 lambda window on the layout space in portrait mode, and 176 by 119 lambda in the landscape mode. These dimensions may be altered by using the <Set Mag> command, which is essentially a zoom feature. The user can select magnifications from 8 times to 1/10 times the default window dimensions. In other words, selecting a magnification of eight times will result in a video display window of 17 by 19 lambda. If a magnification of 1/4 or less is selected, the grid will be automatically turned off. The user can turn the grid back on by using the <Grid On> command. Also, the user can use the <Set Grid> command to select the grid spacing in terms of every n^{th} lambda. A user may delete the contents of some or all layers in the layout space by using the <Zero> command. When selected, the user will be prompted for the layers to be zeroed by using the same technique that the <Set Visible> command uses. The user adds or deleted layers from the currently visible set of layers. Once the user selects the continue response, all layers which are designated visible are deleted.

3.2.2 CifDraw Implementation

The program was designed to take advantage of both the local processing power and the memory of the Concept. Specifically, the program code and layout description is memory resident while the program is executing. No attempt has been made to implement a virtual memory scheme though the data structures can be easily modified to do so. At one point, the Department had two versions of the Concept. Some had 256K bytes of memory, while others had 512K bytes. To address this point, and with a view to the future, the program was designed to use dynamically alterable data structures. The only limitation on the design size is the amount of available memory. This means that a design, which has information on various layers, is limited by the amount of memory, not by the number of layers, as rectangles are retained on the appropriate layer list of the cell pointer as described below. The layout space has been divided into a number of compartments or cells. Each

compartment contains a pointer to data structures which contain rectangles whose center points lie in that compartment. This permits a rapid screen redraw in that one only has to determine which other compartments have rectangles which intersect the current display window. Once this has been accomplished, the graphics library is relied on to clip any objects not falling in the current window.

The following diagram illustrates the major data structures used to retain the layout space information. It should be noted that the Core or GKS notion of a segment has not been implemented or used - the layout space data structures themselves are used to draw the graphic. This represents a 50% memory saving unless the layout space information was strictly file based. In this case, the only objects in volatile memory would be the objects retained and managed in the currently active graphic segments.

Figure 8 - CifDraw Internal Data Structures

Each compartment is 160 lambda (real world lambda) square corresponding to an internal 360 lambda square. Internally, coordinates are scaled by a factor of two as CIF code is specified in integer dimensions, and odd lengths result in non-integer dimensions when the

rectangle is specified using opposite corners. The following diagram illustrates this rather obvious point:

Figure 9 - Unmodified Internal Coordinate Space

However, if all user specified dimensions and coordinates are scaled up by a factor of two, this problem disappears, as is illustrated by the following diagram:

Figure 10 - Modified Internal Coordinate Space

As a scale factor can be applied to an instantiated symbol, this internal scaling is of little consequence to the user. CifDraw users equate one CIF unit to one micron, while Mead and Conway specify one CIF unit as being 1/100 of the micron. Thus, a scale factor of 2/100, when applied to an instantiated symbol, results in integer coordinates. Each compartment contains all rectangles whose center lies within the coordinate span of that compartment. The compartment address for a rectangle is calculated based on the coordinates of the center of

the rectangle. The following calculates the compartment address:

$X_Addr := X_Center \text{ Div } 320$

$Y_Addr := Y_Center \text{ Div } 320$

Where X_Center and Y_Center correspond to the X and Y coordinate respectively of the center of the rectangle in internal dimensions.

This compartmentalization scheme results in a quick method for accessing all rectangles which fall within the currently visible area of the layout space. As situations arise, (long metal tracks), where a rectangle may fall across many compartments, bounds are retained for all compartments expressing the compartment extents (width and height). Using this extent information, the screen refresh procedure can easily determine if a compartment is a candidate for the current screen redraw.

As figure 8 illustrates, dynamic data structures, (structures which can be allocated at run time), are used to retain the layout space information. Each compartment may have a pointer to a first level data structure which contains the following information: the compartment's extents, the set of all layers having rectangles, and the number of rectangles present on each layer with a corresponding pointer to a second-level data structure. The second level data structures, which contain the actual rectangle information, are also dynamically allocated. As figure 8 indicates, each rectangle block can contain up to ten rectangles, along with a pointer to a next block of rectangle information, if there are more present on the same layer.

When rectangles are deleted from a compartment, resulting in an entry rectangle block, that block is attached to a free list as the Pascal procedure `Dispose` is not implemented. Subsequent rectangle block allocations are satisfied from this free list if there are any blocks present, before the Pascal `New` procedure is called.

Four variables are maintained, (`WindowLeft`, `WindowRight`, `WindowTop`, and `WindowBottom`), correspond in internal lambda units to the area of the layout currently being displayed. When advancing the cursor, if the resulting coordinate falls outside these bounds and there is layout space past the bound, then the screen is ~~erased~~, the four variables are updated, and then the screen is redrawn. In one of ~~the early~~ implementations, the screen was scrolled in the appropriate direction when a screen boundary was hit, but was much slower in execution than erasing and redrawing the screen.

The program is modular in design, and was implemented using a top down design, and a bottom up implementation. Program initialization is done by the Prologue routine,

where initial values for global data are established. The default command label annotation which is displayed on the screen, (which corresponds to figure 7), is also initialized in this procedure by calling the Set_Menu routine. Once the program initialization has completed, the program enters a fetch token - execute command loop, (which is a repeat until finished case statement in the Script routine), until the user requests program termination. For every token that is recognized, there is a corresponding statement label in the Script procedure, which contains the names of the procedures to be called for these tokens. Tokens which are not recognized cause program execution to be transferred to the Otherwise label found in this same case statement. If this happens, then the Concept's bell is rung to signify that the user has supplied invalid data. Tokens are returned by the Decode function, which assigns a unique number to each of the forty function keys and four cursor keys. If any other key is hit, then the Decode function returns -1, which is not recognized as a valid token. The reader will note a correspondance between the function key annotation label, the function key token to be returned when that function is requested, and the labels of the case statement found in the Script routine. For example, the tokens CUS_F04 and CSH_F04, which correspond to the <Set SlowSlew> command, are returned by the Concept label management software when <Command-Shift-F4> or <Command-Unshift-F4> respectively is entered by the user. This function key -token association is established by the Set_Menu routine. When Decode returns either of the tokens CSH_F04 or CUS_F04, the Script procedure invokes the routine Set_Slow_Slew, which corresponds to the function key annotation labels.

The procedures which the Script routine calls may be catagorized into two groups - those that act on the layout environment, and those that modify the layout space data. The first group of procedures may be further broken down into those that are primary commands, and those that lead to secondary menus, where a selection from a range of possible values is to be made. The primary commands include: CursorUp, CursorDown, CursorRight, CursorLeft, New_Screen, On_Grid, Off_Grid, Slow_Slew, Fast_Slew, and Slew_To. These procedures are envoked without further intervention by the user when the token corresponding to the user command is received. The commands leading to secondary menus include: Set_Current, Set_Visiblility, Set_Location, Set_Magnification, Set_Slow_Slew, Set_Fast_Slew, Set_Junp_Screen, Set_Grid_Spacing, Plot, and ListDirectory. When any of these procedures are envoked, the user will be required to provide further input. With the exception of ListDirectory and Set_Location, all the user input will be solicited from the function keys, where all the possible valid command input will have replaced the default label

key annotation as indicated in figure 7. Set_Location and ListDirectory prompt the user for keyboard data. When all these procedures which lead to secondary menus have completed executing, the Set_Menu routine is invoked to install the label key annotation as indicated in figure 7.

The second group of procedures include: Mark_To, Delete, Zero, Read_File, and Write_File. These procedures, (with the exception of Write_File), all modify the layout space information. Mark_To creates a rectangle on the currently selected layer when the <Mark> command token is received by the Script routine. The actual rectangle insertion is performed by the Insert_Object function. The reader will note the data abstraction - the fundamental procedures deal with rectangles, not the data structures which retain them. The Delete_Object function does the reverse of the Insert_Object function. Delete_Object is used by the Delete and Zero procedures which correspond to the user commands <Delete> and <Zero>. The Delete function is more complex as it has to locate the rectangle to be deleted before it can actually be removed. If, after deleting the rectangle, a rectangle block becomes empty, then this block is attached to the a free list of empty blocks. This is done as the Pascal procedure Dispose is not implemented. The Read_File procedure prompts the user for the CIF symbol file name which is to be read into the layout space. This file, if it exists, is read in to the layout space relative to the location of the dynamic cursor coordinate at the time of the procedure call. Read_File calls Insert_Object each time it encounters a rectangle command in the CIF symbol file. The Zero procedure prompts the user for the layers to be deleted. Then, for every layer to be deleted, the procedure attaches all non-empty rectangle blocks for all compartments to the free list of rectangle blocks previously mentioned.

When program termination is requested, the Epilogue procedure is invoked, which prompts the user for the name of a file to save the current layout space definition in if it has not been previously saved. Then, the graphics system is terminated which sets the Concept screen back to text mode from graphics mode, the video screen is erased, and the program exits to the operating system.

As the reader will note, adding new capabilities to the program is quite easy. The programmer must modify the Set_Menu procedure to place the feature command name in the appropriate location in the function key annotation area. Then, the corresponding command token is used as the case statement label in the Script procedure. The programmer then associates the program statements, (which is usually the name of a procedure), which are to be executed when the command token is received, with this token label.

Due to the Pascal compiler code segment size restriction previously mentioned, the program has been segmented. To prevent segments from being unloaded and loaded into memory as procedures within the segment become passive, or active, they have been locked into memory. This was accomplished by using a technique known as a "deadly embrace" in which at least one routine in each segment is active for the duration of program execution. This results in increased execution speed of routines not in the next segment, and reduced network traffic to the disk server. It should be noted that the graphics library is also segmented, and is locked into memory once it is initialized.

Chapter four describes how CifDraw has been utilized within the Department. Appendices A-1 and A-2 contain the software listings for the CifDraw program.

3.3 Circuit Simulation

SPICE[9], (Simulation Program with Integrated Circuit Emphasis), is a general purpose analog simulation program for electrical circuits, and originates from the University of California at Berkeley. The program offers three modes of analysis - nonlinear, DC, nonlinear transient, and linear AC. Circuit elements include: resistors, capacitors, inductors, mutual inductors, independent voltage and current sources, four dependent sources (VCVS, VCCS, CCCS, and CCVS), transmission lines, diodes, and transistors (BJT's, JFET's and MOSFET's). The program produces line printer style wave form plots and tabular output. Subcircuits are also supported.

One year after the Concepts were initially purchased, and CifDraw had been implemented, SPICE was converted to run on the machines. Due to the tutorial nature of the environment, and the fall back position the mainframe held, this conversion activity was time well spent. Users can simulate simpler circuits like CMOS operational amplifiers in a reasonable length of time, which is consistent with the goals of the project.

The SPICE conversion was not without difficulty. Due to the program's length, (16000 lines of Fortran), and restrictions imposed by the Concept's editor (about 3000 lines of Fortran per file), editing was a chore. Once the obvious Fortran incompatible code was translated, and the program had been segmented so that no code segment exceeded 32K bytes segment code size restriction, the debugging commenced. SPICE has a built in memory manager system to manage an internal array to optimize memory use. This management system relies on the existence of a function which will return the address of a variable in terms of an offset from the above mentioned internally managed array. On most

machines, this is not a problem as all data is word aligned, which is not the case of the Concept. Thus, it took considerable time and effort before the memory management aspect of the program was operational. The part of the solution involved linking the program in a particular order, which resulted in appropriately aligned data.

To reduce network loading, the program has been overlaid such that the use of MOSFET's only results in completely memory resident code. Users requiring BJT's and/or diodes will notice longer simulation times as the current Concept configuration of 512K bytes of memory is not sufficient to load the complete set of code segments in memory resulting in segment swapping from the disk server.

The following transient analysis statistics for the simulation of the operational amplifier shown in figure 11 provide insight into the suitability of SPICE in a tutorial environment such as the one offered by the Department of Electronics.

Figure 11 - Nine Transistor CMOS Operational Amplifier

Simulation Statistics

	<u>Concept</u>	<u>Mainframe</u>
CPU Time Utilization	671 sec	23 sec
Elapsed Time	671 sec	485 sec

As the table shows, the SPICE transient analysis of the op-amp required 671 seconds of processor time to execute on the Concept. This same simulation required 23 seconds of CPU time on the University's mainframe, which corresponded to 485 seconds of elapsed time, (during a time when the machine was loaded with about 110 users). In terms of the user, (i.e. elapsed time), the Concept required about 1.3 times as much time as the mainframe - a factor which is quite tolerable, even though the Concept is approximately 30 times slower than the mainframe in terms of CPU power. It is to be recognized, as previously stated, that there will always be a requirement for mainframes when large scale simulations are to be run, but the simulation of design building blocks (op-amps etc.) such as those encountered by students and researchers are adequately met. The obvious advantages of the tutorial environment is the accessibility of computing resources. Also, though the Concept may appear to be slower than the mainframe, it is consistently slow regardless of the load. This is not the case for the mainframe.

3.4 Logic Simulation

To address digital simulation, SPLICE[10], a mixed mode simulation program from the University of California at Berkely was ported to the Concept. SPLICE is a hybrid simulation tool offering both electrical and logical simulation in the form of nonlinear time domain transient analysis. SPLICE is an event driven simulator which exploits circuit inactivity to accelerate the simulation. Sections of the circuit which are inactive are not scheduled to be evaluated until signals in the active sections of the circuit cause a signal to change in those inactive sections. Intended for VLSI simulations, the program uses MOS transistors for electrical analysis. As opposed to traditional electrical simulation packages which employ matrix inversion to derive the circuit response, SPLICE uses iterated timing analysis[11], (a relaxation technique), and Gauss-Sidel approximation techniques to solve the circuit response. As a result of this approach, the simulation is one to three orders of magnitude faster than SPICE for an equivalent simulation. The program offers traditional logic gates, (and, or, nor, xor, etc.) signal converters (logic-to-voltage and voltage-to-logic), resistors, capacitors, and n and p channel transistors. The program produces three types of

text file output - waveform, state and tables.

As the program is partitioned into three stand-alone modules: BLT - simulation file preprocessor and syntax checker, SPLICE - the simulation engine, and SPLOT - the output post processor, the execution speed is influenced by the network speed and loading as the three programs are not memory resident for the duration of the simulation run. BLT translates the simulation language input into an intermediate form for the simulation engine. Once BLT has finished executing, SPLICE is called with the intermediate file as input, and produces as output, another intermediate file suitable for input to SPLOT. SPLOT is then called to create the plot file from this intermediate file. Every time a program is executed, it has to be loaded into the Concept's memory from the network disk server. Obviously, network loading will have a significant effect on program execution speed. The following design illustrates the data flow through the SPLICE suite:

Figure 12 - Data Flow For SPLICE Simulations

The following simulation statistics for the circuit shown in figure 13 reveal that SPLICE is in fact faster than SPICE.

Figure 13 - Boot Strapped Inverter

Simulation Statistics

Corvus Concept CPU Time Utilization

-SPICE	- 456 seconds
-SPLICE	- 138 seconds (total)
	85 seconds (simulation)

SPLICE Speed Advantages over SPLICE

- 3.3 (total)
- 5.4 (simulation)

As the simulation statistics suggest, for time domain transient analysis, SPLICE has a definite speed advantage over SPICE, though not the order of magnitude speed advantage claimed by the authors. This reduced speed advantage may be attributed to the small number of nodes in the circuit that was simulated.

As the next chapter describes, SPLICE is used extensively to complement the hardware component of a final year undergraduate course (97:450). Students simulate portions of circuits which they later breadboard, drawing conclusions and comparisons between the simulation and hardware results.

3.5 Filter Analysis

3.5.0 Introduction

The programming environment described in this thesis lends itself to CAD tool design and development. Using this programming environment, new tools can be provided to augment evolving curricula. This section describes one such development. An analog electronics course instructor (Professor M. A. Copeland) felt that it would be desirable to offer a CAD tool which would be suitable to support the design of continuous filters. The COMPLEX program, written in Pascal and Fortran, was designed and implemented to meet this need. This program allows students to plot the magnitude and phase response of continuous filters, composed of cascaded biquadratic sections, on the video display of the Concept computer. COMPLEX is used by the students to tune their filter designs in order to meet tightly specified magnitude responses. These tuning parameters are then used to establish the initial values for the hardware realization of their filter design, using standard architectures. Once the hardware design has been successfully tuned to meet the design specifications, the students can enter these final design parameters into the COMPLEX program to compare the response to the initial values. The program accepts the tuning parameters as H's, Ω 's and Q's for each stage. As a convenience, all program input and output is specified in Hertz. Once the appropriate values are entered, a plot of magnitude and phase response versus logarithmic frequency can be requested. The program also provides for hardcopy output of the response information in a tabular form, and as a plot. The following sections describe the COMPLEX program.

3.5.1 The COMPLEX Program

Complex is a program which can be used to plot the magnitude and phase response of continuous filters composed of biquadratic stages. The program uses the following equation, which is a slightly modified version of equation 5.87 found in [15], which describes the general form of a biquadratic transfer function

$$T_1(s) = H_{01}(k_{11}s^2 + k_{21}(\Omega_{0z1} - Q_{z1}s + k_{31}\Omega_{0z1}^2) / (s^2 + (\Omega_{0p1} - Q_{p1})s + \Omega_{0p1}^2)$$

where

T_1	is the response of stage 1
H_{01}	is the gain factor of stage 1
k_{j1}	is the value of k_j for stage 1
Ω_{0z1}	is the value of the zero frequency for stage 1

Ω_{opi} is the value of the pole frequency for stage i
 Q_{Ozi} is the value of Q for the numerator of stage i
 Q_{opi} is the value of Q for the denominator of stage i
 s is the Laplace variable

The table below, which is based on table 5.1 found in [15], identifies five different response characteristics

Coefficient			Filter
k_1	k_2	k_3	Characteristic
0	0	1	Lowpass
1	0	0	Highpass
0	1	0	Bandpass
1	0	1	Bandstop
1	-1	1	Allpass

All program input and output is specified in terms of Hertz. The user may specify up to ten active stages. For each active stage, the values of the coefficients found in the above equation are to be specified. The user may then specify the frequency bounds for the analysis. The program, when requested, will plot the magnitude response (in dB) and the phase response (in degrees) along a logarithmic frequency axis on either the video display unit, or on a hardcopy plotter. The program provides a file option for loading and saving the coefficient information. Also provided is a feature to generate a file containing a table of the magnitude and phase response against either logarithmic or linear frequency. The following sections describes the function of each of the COMPLEX commands.

Command	Description
SetStage	Displays the value of the currently selected stage, and then prompts the user for the stage number to which all further parameter modifications (Ω 's, H 's, k 's, and Q 's) will be made. Valid input is an integer in the range from one(1) to ten(10) inclusive.
Set # Of Stages	Displays the value of the currently selected stage number, and then prompts the user for number of stages to be used for the analysis.
$k_1(i)$	Displays the value of k_1 for the currently selected stage (i), and then prompts the user for a new value

$k_2 (1)$	Displays the value of k_2 for the currently selected stage (1), and then prompts the user for a new value
$k_3 (1)$	Displays the value of k_3 for the currently selected stage (1), and then prompts the user for a new value
$W_{02} (1)$	Displays the value of Ω_{02} for the currently selected stage (1), and then prompts the user for a new value
$Q_{02} (1)$	Displays the value of Q_{02} for the currently selected stage (1), and then prompts the user for a new value
$W_{0p} (1)$	Displays the value of Ω_{0p} for the currently selected stage (1), and then prompts the user for a new value
$Q_{0p} (1)$	Displays the value of Q_{0p} for the currently selected stage (1), and then prompts the user for a new value
$H_0 (1)$	Displays the value of the gain factor for the currently selected stage (1), and then prompts the user for a new value
Set Low Frequency	Displays the value of the lower frequency bound, and then prompts the user for a new value. The default value for this low frequency bound is one Hertz.
Set High Frequency	Displays the value of the upper frequency bound, and then prompts the user for a new value. The default value for this upper frequency bound is 10,000 Hertz.
Display Coeffs	Prints a table of the current coefficients values (Ω 's, H 's, k 's, and Q 's) on the video display unit.
Concept Plot	Plots a magnitude (in dB) and phase (in degrees) response against logarithmic frequency for the system currently defined, on the Concept video display unit. Two hundred points in frequency are used for the analysis.
LA120 Plot	Produces plot file of magnitude and phase response against logarithmic frequency which is compatible with an LA120 printer equipped with a Selanar graphics board.
HP7721 Plot	Plots a magnitude and phase response against logarithmic frequency for the system currently defined, on a locally attached HP7721 plotter.
WriteLin Response	Creates a file called RESPONSE.TEXT in the current default directory which contains a table of magnitude and phase response against linear frequency. Two hundred points

equally spaced along a linear frequency axis are used for the analysis

WriteLog Response	Creates a file called RESPONSE.TEXT in the current default directory which contains a table of magnitude and phase against logarithmic frequency. Two hundred points equally spaced along a logarithmic frequency axis are used for the analysis
LoadFile	Loads a previously saved file called COEFFS.TEXT, containing the values for the Ω 's, Q's, H's, and k's
SaveFile	Saves the current values of the Ω 's, Q's, H's, and k's in a file called COEFFS.TEXT
Exit	Saves the current values of the coefficients in the file COEFFS.TEXT before program termination

- Appendices C-1 and C-2 contain the software listings for the COMPLEX program

Chapter 4 - Design Environment Utilization

4.0 Introduction

The CAD environment developed in this thesis has been used at varying stages of refinement for three years now, for both education and research. This chapter describes how the system has been tailored for and used for both education (undergraduate and graduate courses) and research (graduate thesis work).

4.1 User Accounts

To a user, the Concept appears as a terminal connected to a computer, once the machine has executed its power-up diagnostic sequence. The user is solicited for an account name and a password, before machine access is granted. Due to the number of students, and the transient nature of their use, a scheme of machine specific accounts was implemented, with each machine being given a unique account name. This scheme results in a system where anyone can use any machine, and accounts do not have to be created for new users - they are simply instructed as to the logon procedure. This vastly reduces the administration requirements for the System Administrator. To prevent users logging into machines for which the account name was not authorized, at logon, a program checks the account name against an authorization table of account names and machine identifications. Each machine has a unique identification, required for the local area network operation. If the account name is not authorized for the logon machine, the user is informed, and the machine reboots.

The vendor supplied network disk servers have three problems:

1. The disk servers do not ensure that multiple writers are granted exclusive disk access.
2. The disk servers are contiguous block-structured devices
3. A disk volume can only contain 77 files

If the disk-servers were non-contiguous record oriented devices, and did not have a maximum limit of 77 files per disk volume, then a single account would be sufficient for all users involved in similar activities.

Also associated with each machine account is a 1500 block (768k byte) file storage volume on the network disk server. This volume is for the exclusive use by the user for the duration of the logon session. Any files created in this volume may exist until the next time the user returns, but there is no guarantee of this. Users are encouraged to retain copies of

files in 8" floppy disks. To address this mode of operation, one machine in each machine room location is equipped with a floppy disk drive. Any user's files to be copied to the hard disk are moved into a temporary volume on the same disk server. The user then goes to the machine to be used for the session and copies the appropriate files from this temporary volume to the machine specific volume. At the conclusion of the session, any files to be saved are copied to the temporary volume, then the user proceeds to the floppy equipped machine to transfer the files to his personal disk.

Users who are involved with long term or large projects are generally granted a permanent area on the disk, if space permits. As the disk is block structured, this does lead to problems in situations where all allocated volumes are not the same size. Specifically, disk fragmentation occurs requiring that the active values be coalesced to free up the otherwise unusable disk space. This process requires considerable effort by the system maintainers, leading to a reluctance to provide users with permanent disk space.

4.2 Network Consolidation

Initially, the environment was partitioned into two electrically distinct networks. During the development phase of the project, this led to eventual problems with network printing and software version control. Initially, an Apple II computer with two Omninet[12] transporter cards, (one for each network), was used to alternatively despool from each network. Hardware problems with the Apple II caused problems with areas of the hard disks which are used to implement semaphores. This area was not accessible via the vendor supplied software, requiring that the hard disk be reformatted. Also, by having these two distinct networks, users were required to physically move the flatbed plotter to a machine connected to the network which they were using. As this meant that there were occasions where the plotter would be left in an insecure area, a consensus dictated that for the previously mentioned reasons, the two networks should be consolidated into one.

In order to avoid a total critical point failure, each of the two network disk servers are partitioned as two logically distinct network disks with approximately the same number of users on each. At the expense of disk space utilization, each disk server has a complete complement of system and application software, so that in the event of a server failure, only half of the users will be affected. The system has been set up in such a manner that if one logical network is going to be unavailable for any length of time, then the affected users can be transferred to the functioning disk if necessary, and space provided.

4.3 The Network

Consolidating the two networks has had an obvious effect on network speed. As previously mentioned, the networking technology is base-band RS422 operating at one million bits per second. There are approximately twenty network nodes, of which fifteen are likely to be active at any given time. This leads to frequent network collisions, some of which are fatal due to the vendor's [12] network interface, which is carrier sense multiple access with random restart (as opposed to the Ethernet CSMA/DC approach). The network transporters give up after ten unsuccessful attempts at accessing the network disk. This is not unusual when five or more network machines are attempting to boot at once as the Concept operating system is moderately large (approximately 50K bytes of code, of which all must be resident before booting commences past the hardware confidence test).

To address the inherent speed limitation imposed by the network, all software developed at Carleton is memory resident for both data and code, where possible. Thus, aside from initially loading the program into memory, possibly with a data file, the network is not accessed again until the program terminates, or a file is completely read or written. In the case of large programs like SPICE, which is approximately 400K bytes of code, it is necessary to overlay it, requiring a network access when the currently executing overlay is finished, and the new overlay is being loaded. The operating system can be fooled into locking overlay segments into memory, and this feature has been used in all programs developed at Carleton. Without doing so, even though there may be adequate memory to completely load the program to be executed and all its overlay segments, the loader only loads segments as needed, and discards them once all routines in the segment are inactive.

4.4 Educational Use

4.4.1 Graduate Use

The first user of the CAD environment was a limited enrollment graduate level course called "Integrated Circuit Engineering II". Each member (there were 3) of the class designed a different type of analog to digital converter, which became the basis of the Department's first submission to the newly formed Canadian Microelectronic Corporation (CMC). The CMC is a vehicle offering CMOS and NMOS silicon foundry services to Canadian Universities. Each of these three CMOS designs were laid out using the CifDraw program and then sent to the University's mainframe computer, (using the Export program found in Appendix 3), where the design files were copied to nine track magnetic tape. The tape was

then sent to the CMC via Queen's University for integration into multiproject chips in preparation for mask making.

During this first course offering, the simulation facilities described herein were not available, and all electrical simulations were done using SPICE on the university mainframe. One of these designs, a successive approximation analog-to-digital converter, was tested in a 4th year project this past year. This shows a difficulty with the logistics of university VLSI design courses. Due to the length of time required for design and fabrication, the original designers are rarely around to actually test their circuits. Rather than have the circuits 100% untested, other students are given the task of establishing the circuits operation in order to provide future designers with process information.

The next course offering involved more students and bigger projects. A digital voltmeter was the basis for the design of two large IC's, which were sent to the CMC for fabrication. The class was divided into two groups, one responsible for the analog portion, and the other responsible for the digital portion of the design. SPICE had just become available in the design environment, and was used for building block scale simulations as previously described. SPLICE was not to be available until the third course offering. Due to the dated version of SPLICE running on the university's mainframe, this program has not been used by students in the Department of Electronics until it was made available in the design environment described in the thesis.

During the third course offering, a group of graduate students used the CAD design environment to design CMOS standard cells. These standard cells were then used as the building blocks by other designers. Initially, the cells were designed, simulated, and laid out on the Concepts. Then, when the industrial quality CAD workstation (Metheus) arrived, these standard cells were transferred to it, to be integrated into the rich design environment found there. The Metheus provides resources and design tools which are unavailable on the Concepts. The Concepts connected to the Metheus workstation illustrate the effectiveness of a layered design environment. Students learn the basics of IC design using the Concept network. Work done here may form the basis for more advanced research which is carried out on the Metheus, where there are generally fewer students. IC designs developed on the Metheus are quite often sent to the CMC for fabrication, while those done on the Concepts are more likely to be fabricated within the Department, or form the building blocks for the Metheus based designs.

4.4.2 Undergraduate Use

Possibly one of the most successful design courses to use the CAD environment was 97.469 - Integrated Circuit Design and Fabrication - taught by Professor G. Tarr[14]. Otherwise known as the two week NMOS IC course, students designed simple circuits for fabrication using the Department's NMOS process. A Concept connected to a Mann pattern generator was used to produce the masks directly from the student's CIF files, located on the network hard disk. Once the circuits had been fabricated, the students wafer probed the results to establish their operational characteristics. This design experience and access to an actual fabrication process has provided the fourth year students with a resource which has traditionally been reserved for researchers - the mentor based education process of integrated circuit design within industry is no longer so necessary.

SPLICE, the logic simulation program, has been heavily used in 97.450 - Electronic Circuit and System Design. Here, students were expected to simulate one of the building blocks (the clock generation chain) of a digital voltmeter, which they then constructed on breadboards. This provided the students with an experience which closely parallels the industrial design process. The industrial design process involves the simulation and circuit layout of a design. Sometimes circuit ideas are actually breadboarded to demonstrate their correctness. The student's use of the design environment provides valuable experience in that the fundamental design tool - SPLICE - was intended for large scale MOS designs, and is a tool they might actually encounter in their future endeavours. It is also important for today's engineering graduates to learn and understand that electronic circuits and systems can largely be designed without breadboarding. A previous offering of this course, (97.450), required that students design a continuous filter to meet tight specifications. The students used their preliminary design calculations as the starting parameters for the COMPLEX program described in chapter 3. Once their frequency responses satisfied the required specifications, the students then used these tuned parameters to arrive at initial component values for biquad filter stages. They then tuned their hardware filters, and then compared the final responses to those initially determined with the COMPLEX program.

The author personally taught 97.476 - Digital Integrated Electronics - where the students had to debug the control signal generator for a DRAM controller, before they actually built it. The students were provided with a synchronous DRAM controller circuit which had a few intentional bugs in it. They were also provided with some timing information on the required signals to control the memory device (a Motorola 6665 dynamic

memory), along with some philosophy on synchronous circuit design. From this information, they were expected to locate and eliminate the trivial bugs, and derive the appropriate timing information by using SPLICE to simulate circuit operation. The simulation results were then used to verify the operation of their working DRAM systems, once the circuits had been breadboarded.

4 The successful integration of the tutorial CAD environment into the curriculum has demonstrated the usefulness and appropriateness of such an attempt. Students are expected to work at a much more complex level now, requiring powerful tools like those described here. In the fall term, students are provided with a handout, (see appendix D), which describes the system and how to use it. When required, they either purchase, or are provided with, manuals which describe the operation of the simulators. Armed with this information, they proceed through their fourth year with little difficulty.

Chapter 5 Conclusions

5.1 Summary

The work described in the previous chapters of the thesis has been used to create a distributed design environment as illustrated below

Figure 14 - CAD Network

As the reader will note, the network forms the backbone of the system, providing disk storage and hardcopy output facilities for the network users. From the figure above, it is apparent that there are three major users of the system. The block of Corvus Concepts to the left of the network are primarily used for undergraduate IC design, introductory graduate IC design, and graduate level standard cell design. The block at the top on the right which

includes the Metheus workstation represents the migratory path that the previously mentioned graduate student standard cell designs take to the more powerful and expensive Metheus workstation. The block below this indicates that the Department's mask making facilities have been integrated into the design environment.

Software based tools for circuit layout and analysis have been offered for IC design education. Students can use the simulators discussed in the thesis for gate and device level simulations. The layout editor can be used for geometrically describing an IC, which can then be fabricated locally in the Department's nMOS process, or remotely via the CMC for fabrication in CMOS.

5.2 Future Work

5.2.0 Introduction

The design environment presented in the thesis represents the foundation upon which further work can be built. Over the past three years, areas for development and improvement have been identified and noted. These areas may be categorized as follows:

1. Hardware Improvements
2. Software Improvements
3. Future Machines

Each of these categories will be discussed below.

5.2.1 Hardware Improvements

Since all the Concepts are connected to one network, a degradation in speed performance has been observed, especially when executing large programs. This performance degradation will be noticed by all network users trying to access the disk servers when such a large program is being transferred across the network to a target host. This is attributed to the rather low network speed (by today's standards) of 1M bits per second. It takes approximately 3.7 seconds to load SPICE into a Concept from a network attached disk server, with no other network activity. During a simulation laboratory, when many students are iterating between a text editor and a simulation program while debugging their simulation exercises, the amount of time required to load programs from the network disks into the computers is considerably longer.

This program access speed performance can be significantly improved by equipping the Concepts with RAM disks. A RAM disk is constructed with standard memory devices, but appears as a disk drive to the host machine, and consequently, RAM disk accesses are as

fast as the memory devices used. Using a RAM disk is economically feasible as the cost of memory is quite inexpensive (approximately 0.02 cents per byte for 256K bit chips). To the Concept, the RAM disk would appear as a hard disk based volume, which would be located in the memory within the address space of the executing machine. A user could then store all the files relevant to his needs on the RAM disk by copying the appropriate files from the disk server at the start of the session. All future references to required files and programs would be satisfied from this semiconductor disk. The RAM disk could be installed in two ways: 1) replace the existing 64K bit chips with 256K bit chips, and 2) use an existing Apple II compatible RAM disk card in one of the Apple compatible peripheral slots (there are four) on the Concept's processor board. Both approaches require that a RAM disk emulation program be developed. The vendor has already provided software to read Apple II disks via the peripheral slots, so this would be within the abilities of an experienced programmer.

The first approach would require that the memory board addressing multiplexers be appropriately modified to account for the four-fold increase in memory chip size. The manufacturer in fact designed the memory board with these larger chips in mind. This approach would be quite inexpensive once a machine using the 256K bit chips was modified and debugged. The second approach would be considerably easier as RAM disk cards are available off the shelf for Apple II computers, but this approach is considerably more expensive due to the extra support circuitry and development costs.

The Concept's floating point instruction performance could be improved. This is particularly important for numerically intensive programs such as SPICE. The author supervised an undergraduate research project, where the student (Graham Bell) investigated interfacing an existing Apple II compatible floating point processor to the Concept. The results of the study were disappointing but understandable. The floating point processor used did not support the IEEE floating point format, so considerable processing time was wasted converting numerical formats between the Concept and the numerical processor. Also, because the vendor's high level language libraries were provided in object form only, the definitions for floating point operations could not be redefined to use the floating point processor. If time were spent interfacing an IEEE compatible floating point processor to the Concept, and if the vendor will provide software listings for the language libraries, then one can expect to see a considerable execution speed improvement in SPICE.

Appendix C - COMPLEX Filter Analysis Program Software Listings

1. Complex This file contains the complete listing for the COMPLEX program described in Chapter 3.
2. Plot This file contains the software listings for a logarithmic plotting routine which is called by COMPLEX (or any other Fortran program).
3. Export This file contains the software listings for a program which is used to send files to the University's mainframe.

Appendix D - Handbook as seen by Students

SPLICE currently provides low resolution lineprinter quality plots, where a 200 time step plot requires 200 lines, which equates to 6 pages of paper including the header and trailer pages. The graphics post processor will be able to plot the complete simulation results on one page for the same number of time steps.

A better file transfer mechanism is required between the Concept computers and the other machines to which they are often connected. As mentioned in chapter 4, a program (Export) was written to send files to the University's mainframe, but this is a very specific solution for a more general problem. Within the university community, the Kermit file transfer protocol from Columbia University in New York is becoming the defacto standard. This is due to the fact that it is a public domain program which is supported by Columbia. Within the Department, Kermit is currently running on the Vax, the Metheus, and a number of IBM compatible personal computers. Time should be spent developing a Kermit compatible file transfer program for the Concept, to allow it to be connected to the rest of the Department's computers.

One of the author's goals, when the CAD environment development commenced, was to create a unified access mechanism to the network from the various computerized instruments within the Department of Electronics. To some degree, this goal was met. A fourth year project under the author's supervision successfully linked a computerized semiconductor measurement (CSM) system to the network. This allowed researchers to transmit data gathered on the CSM machine to the network connected disks. From there, it could be accessed by the Concept computers for further analysis. Currently, semiconductor process parameter extraction is not automated in terms of parameters for SPICE simulations, but it could be. Also, if the VLSI test station were connected to the network, then researchers could automatically correlate simulations with actual measurements.

5.2.3 Future Machines

Chapter 2 discussed the Apple Macintosh as a possible candidate for a contemporary machine suitable for an IC design environment for education. The user interface makes the machine amenable for novice use. This, coupled with the fact that many American universities are members of the Apple University Consortium, (Apple Computer offers discounts of up to 40% discounts to member universities), suggests that the Macintosh will command a high profile in post secondary institutions. A review of personal computing journals reveals that many young software companies are developing electronics engineering

CAD tools for the Macintosh (IC layout, schematic editors, netlist extractors). The author in fact began porting SPICE to the Macintosh six months ago.

5.3 Conclusions

The goal of this thesis has been satisfied. An environment suitable for the teaching of CAD based IC design has been developed using an inexpensive personal computer. This environment has been used in varying degrees of refinement over the past three years. Using the tools described in the thesis, graduate students have designed LSI CMOS circuits which have been fabricated using the facilities offered by the Canadian Microelectronic Corporation. Undergraduate students have used the environment for the analog simulation component of a fourth year course, as well as for IC design. These designs have been fabricated using the Department's nMOS process. The workstations have also been used to design CMOS standard cells, which have been transferred to the more sophisticated and expensive Metheus workstation.

References

- 1 C. Mead and L. Conway, *Introduction to VLSI Systems*, Menlo Park: Addison-Wesley Publishing Company, 1980.
- 2 *The 8086 Family User's Manual*, Intel Corporation, 1979
- 3 T. Harmon and B. Lawson, *The Motorola MC68000 Microprocessor Family*, Englewood Cliffs: Prentice-Hall, 1985
- 4 K. Jensen and N. Wirth, *Pascal User Manual and Report*, New York: Springer-Verlag, 1974
- 5 W. Newman and R. Sproull, *Principles of Interactive Computer Graphics - Second Edition*, New York: McGraw-Hill Book Company, 1979
- 6 J. Foley and A. VanDam, *Fundamentals of Interactive Computer Graphics*, Reading Massachusetts: Addison-Wesley Publishing Company, 1982
- 7 D. W. Lewis, "CIFSYM Users Manual", EECS Department, University of Santa Clara, Santa Clara, CA, 1982
- 8 *SVS Pascal Manual*, Silicon Valley Software Inc., San Jose, CA
- 9 L. W. Nagel, "SPICE2 - A Computer Program To Simulate Semiconductor Circuits", Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA, 1975.
- 10 R.A. Saleh, J.E. Kleckner, and A.R. Newton, "SPLICE Version 1.7 User's Guide", Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1983.
- 11 R.A. Saleh, J.E. Kleckner, and A.R. Newton, "Iterated Timing Analysis in SPLICE", Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1983.
- 12 *Corvus Omninet Programmer's Guide*, Corvus Systems, Inc., San Jose, CA, 1982.
- 13 *Hewlett-Packard 7221C and 7221T Graphics Plotter Operating and Programming Manual*, Hewlett-Packard, Inc., San Diego, CA, 1981
- 14 N.G. Tarr, C.H. Chan and L.P. Berndt, "Multiproject Chip Fabrication in a University Environment", Carleton University, Ottawa, Canada, 1984
- 15 M.E. Van Valkenberg, *Analog Filter Design*, Holt, Rinehart and Winston, New York, 1980

Bibliography

D. Glendinning and M.A. Copeland, "A Personal Workstation Based Interactive Graphics Editor for IC Layout", 1983 Canadian Conference on Very Large Scale Integration, Waterloo, Ontario.

J. Goldak, N. Holtz and D. Glendinning, "Carleton University's Computer Aided Engineering at Carleton", CAD/CAM - Education in the Next Decade, Algonquin College, 1983.

D. Glendinning and M.A. Copeland, "A Personal Workstation Based Environment for Electrical Engineering", Western Education Computing Conference, San Francisco, 1983.

D. Glendinning, "A 68000 Based CAD Environment", Department of Electronics Seminar, Carleton University, 1983

Appendices

Appendix A - Two-Dimensional Graphics Library Software Listings

1. GraphDefs Graphics package type declarations unit. This file contains the global type definitions required when using the two-dimensional graphics package. Also contained in this unit are two utility functions.
2. Graph Device independent graphics library unit. This file contains the applications programmer interface procedure to the graphics package. Also contained in it are some utility functions which include: integer to character string conversion, real to character string conversion, and routines to control the Concept screen when it is in text mode.
3. IoConcept Concept video screen device driver. This file contains the operating system interface routines for the library.
4. IoLA120 Selanar graphics board device driver. This file contains the device driver for talking to a DecWriter III equipped with a Selanar graphics card. Contained in this file is the complete vector to raster scan conversion program, which utilizes the restartable line drawing algorithm discussed in Chapter 3.
5. IoHP7221 HP7221C plotter device driver. This file contains the device driver for talking to an HP7221 plotter. The data encoding methods for talking to the device are found in [13]. This file also has an example of run-time loading a hardware device driver.
6. IoWindow Concept window interface unit which interfaces to the vendor's supplied system library.
7. Create Program for creating software graphics characters.

Appendix B - CifDraw IC Layout Program Software Listings

1. LayoutDefs This file contains the unit software listing for the CifDraw program. Contained in the file are: constant declarations used for the command dispatcher, statically declared arrays, and character strings; and all global variable declarations.
2. CifDraw This file contains the complete IC layout program listings discussed in Chapter 3.
3. CifBias This file contains the software listings for a program which can be used to transform symbols created by the CifDraw program.

Appendix C - COMPLEX Filter Analysis Program Software Listings

1. Complex This file contains the complete listing for the COMPLEX program described in Chapter 3.
2. Plot This file contains the software listings for a logarithmic plotting routine which is called by COMPLEX (or any other Fortran program).
3. Export This file contains the software listings for a program which is used to send files to the University's mainframe.

Appendix D - Handbook as seen by Students

Appendix A - Two-Dimensional Graphics Library Software Listings

Appendix A-1 - GraphDefs Unit

```

Unit GraphDefs;
Interface
  Type
    ScreenColour = (None, Black, Brown, Blue, Green,
                    Lime, Red, Orange, Yellow, White, Flip);
    LineType      = (Dotted, Dashed, Solid);
    DeviceType    = (Concept, HP7221, LA120, NoDevice);
    String255     = String [255];
  Function Min    (Arg_1, Arg_2: Integer): Integer;
  Function Max    (Arg_1, Arg_2: Integer): Integer;
{EndInterface}
Implementation
  Function Min (Arg_1, Arg_2: Integer): Integer;
  Begin {Min}
    If Arg_1 <= Arg_2
    Then
      Min := Arg_1
    Else
      Min := Arg_2
    {EndIf}
  End; {Min}
  Function Max (Arg_1, Arg_2: Integer): Integer;
  Begin {Max}
    If Arg_1 >= Arg_2
    Then
      Max := Arg_1
    Else
      Max := Arg_2
    {EndIf}
  End; {Max}
{EndImplementation}
End.

```

Appendix A-2 - Graph Unit

```

{SR-} {Turn off range checking.}
{SS Graph}
Unit Graph;
Interface
  Uses
    {SU /ccutil/GraphLib.Obj}
    GraphDefs,
    IoHP7221,
    IoConcept,
    IoLA120;
Function InitGraphics (Device: DeviceType): Boolean;
Procedure InqDevice (Var XMin, XMax, YMin, YMax,
  XDotsPerInch, YDotsPerInch: Integer);
Procedure InqColour (Var Colour: ScreenColour);
Procedure InqStyle (Var Style: LineType);
Procedure SetWindow (XMin, XMax, YMin, YMax: Integer);
Procedure SetViewport (XMin, XMax, YMin, YMax: Real);
Procedure SetDevice (Var XMin, XMax, YMin, YMax: Integer);
Procedure SetColour (Colour: ScreenColour);
Procedure SetLine (Style: LineType);
Procedure SlewAbs (X, Y: Integer);
Procedure SlewRel (X, Y: Integer);
Procedure DrawAbs (X, Y: Integer);
Procedure DrawRel (X, Y: Integer);
Procedure TextAbs (X, Y: Integer; Chars: String255;
  Theta, Slant: Real;
  Width, Height: Integer);
Procedure TextRel (X, Y: Integer; Chars: String255;
  Theta, Slant: Real;
  Width, Height: Integer);
Procedure LineAbs (X1, Y1, X2, Y2: Integer);
Procedure LineRel (X1, Y1, X2, Y2: Integer);
Procedure PointAbs (X, Y: Integer);
Procedure PointRel (X, Y: Integer);
Procedure NewPage;
Procedure TermGraphics;
Procedure Real_To_Str (Arg: Real; Decimals: Integer;
  Var Result: String255);
Procedure LInt_To_Str (Arg: LongInt; Var Result: String255);
Procedure Move_Cursor (X_Position, Y_Position: Integer);
Procedure Erase_Screen;
Procedure Erase_Line;
Procedure Ring_Bell;
Procedure Normal_Video;
Procedure Inverse_Video;
Procedure Curse_On;
Procedure Curse_Off;
{EndInterface}
Implementation
  Const
    Max_Letters = 127;
    Max_Coords = 500;
  Type
    Edge = (Left, Right, Bottom, Top);
    Matrix3x3 = Array [1..3, 1..3] of Real;
    Type_Byte = -128..127;
    Type_Coord = Type_Byte;
  Var
    {Window On The World To Viewport}
    W_To_VP: Matrix3x3;
    {Viewport To Virtual Device}
    VP_To_VD: Matrix3x3;
    {Virtual Device To Physical Device}
    VD_To_PD: Matrix3x3;
    {W_To_PD <- W_To_VP * VP_To_VD * VD_To_PD}
    W_To_PD: Matrix3x3;

```



```

Window:           Array [Edge] Of Integer;
Screen:           Array [Edge] Of Integer;
CurrentX:         Integer;
CurrentY:         Integer;
RelX:             Integer;
RelY:             Integer;
CurrentDevice:    DeviceType;
CurrentColour:    ScreenColour;
CurrentStyle:     LineType;
Char Width:       Integer;
Char Height:      Integer;
Char Spacing:     Integer;
Letters:          String255;
Ptrs:             Array [0..Max Letters] Of Integer;
Coords:           Array [1..Max_Coords, 1..2] Of Type_Coord;

Procedure SegUnloc; External;
Procedure SegLock; External;
Procedure Write Integer (Argument: Integer);
Begin {Write Integer}
  Write (Chr (Argument Div 256), Chr (Argument Mod 256))
End; {Write Integer}
Procedure Write Byte (Argument: Integer);
Begin {Write Byte}
  Write (Chr (Argument Mod 256))
End; {Write Byte}
Procedure Write Escape And (Argument: Char);
Begin {Write Escape And}
  Write (Chr (27), Argument)
End; {Write Escape}
Procedure Write Char (Argument: Char);
Begin {Write Char}
  Write (Argument)
End; {Write Char}
Procedure Move Cursor;
Begin {Move Cursor}
  Write Escape And ("=");
  Write Byte (X Position);
  Write Byte (Y Position)
End; {Move Cursor}
Procedure Erase Screen;
Begin {Erase Screen}
  Write Escape And ("J")
End; {Erase Screen}
Procedure Erase Line;
Begin {Erase Line}
  Write Escape And ("K")
End; {Erase Line}
Procedure Ring Bell;
Const
  Bell = 7;
Begin {Ring Bell}
  Write Byte (Bell);
End; {Ring Bell}
Procedure Inverse Video;
Begin {Inverse Video}
  Write Escape And ("G");
  Write Char ("4")
End; {Inverse Video}
Procedure Normal Video;
Begin {Normal Video}
  Write Escape And ("G");
  Write Char ("0")
End; {Normal Video}
Procedure Curse On;
Begin {Curse On}
  Write Escape And ("c")
End; {Curse On}
Procedure Curse Off;
Begin {Curse Off}
  Write Escape And ("b")

```

```

End; {Curse Off}
Procedure Init_Soft;
Const
  Name_In      =  '/ccutil/Graphic_Letters';
Var
  Next_Loc:      Integer;
  File_In:      Interactive;
Procedure Init;
Begin
  Letters := '';
  Next_Loc := 1;
  Reset (File_In, Name_In);
End;
Procedure New_Letter (Ch: Char);
Begin
  Letters := Concat (Letters, Ch);
  Letters[Length (Letters)] := Ch;
  Ptrs[Length (Letters)] := Next_Loc
End;
Procedure Draw (X, Y: Type_Coord);
Begin
  Coords[Next_Loc, 1] := X;
  Coords[Next_Loc, 2] := Y;
  Next_Loc := Next_Loc + 1
End;
Procedure Slew (X, Y: Type_Coord);
Begin
  Draw (X + Char_Spacing, Y)
End;
Procedure Term;
Begin
  Ptrs [0] := Ptrs [1];
  Ptrs [Length (Letters) + 1] := Next_Loc;
  Close (File_In, Lock);
End;
Procedure Read_Letters;
Var
  Current_Character:  Char;
  Mode:              Char;
  I:                 Integer;
  Count:             Integer;
  X:                 Type_Coord;
  Y:                 Type_Coord;
Begin
  Readln (File_In, Char_Width, Char_Height, Char_Spacing);
  While Not Eof (File_In) Do
    Begin
      Readln (File_In, Current_Character, Count);
      New_Letter (Current_Character);
      For I := 1 To Count Do
        Begin
          Readln (File_In, Mode, X, Y);
          If Mode = 'S'
          Then
            Slew (X, Y)
          Else
            Draw (X, Y)
          {EndIf}
        End
      {EndFor}
    End
  {EndWhile}
End;
Begin
  Init;
  Read_Letters;
  Term
End;
Procedure LInt_to_Str {(Arg: LongInt; Var Result: String255)};
Var

```

```

sign:      Char;
ch:        Char;
x:         Integer;
Begin
  Result := '';
  sign := '-';
  If Arg = 0
  Then
    Begin
      Result := '0';
      Exit (Lint_To_Str)
    End;
  {EndIf}
  If Arg < 0
  Then
    Begin
      If Arg = $80000000
      Then
        Begin
          Result := '-2147483648';
          Exit (Lint_To_Str)
        End;
      {EndIf}
      sign := '-';
      Arg := Abs (Arg)
    End;
  {EndIf}
  While Arg <> 0 Do
    Begin
      x := Arg Mod 10;
      Arg := Arg Div 10;
      ch := Chr (Ord ('0') + x);
      Result := Concat ('', Result);
      Result[1] := ch
    End;
  {EndWhile}
  If Sign <> ''
  Then
    Begin
      Result := Concat ('', Result);
      Result [1] := Sign
    End;
  {EndIf}
End;

Procedure Real_To_Str ((Arg: Real; Decimals: Integer;
  Var Result: String255));
Var
  Scale:      Real;
  Temp:       String255;
Begin
  Scale := PwrOfTen (Decimals);
  If Arg >= 0.0
  Then
    Result := ''
  Else
    Begin
      Result := '-';
      Arg := Abs (Arg)
    End;
  {EndIf}
  Arg := Arg + 1.0 / (2.0 * Scale);
  Lint to Str (Trunc (Arg), Temp);
  Result := Concat (Result, Temp);
  Lint To Str (Trunc (Arg * Scale) Mod Trunc (Scale), Temp);
  Result := Concat (Result, Temp);
End;

Procedure Mat_Mul (Var Result: Matrix3x3;
  First, Second: Matrix3x3);
Var
  I:          Integer;

```

```

J: Integer;
K: Integer;
Temp: Matrix3x3;
Sum: Real;
Begin {Mat Mul}
  For I := 1 To 3 Do
    For K := 1 To 3 Do
      Begin
        Sum := 0.0;
        For J := 1 To 3 Do
          Sum := Sum + First [I,J] * Second [J,K];
        {EndFor}
        Temp [I,K] := Sum
      End;
    {EndFor}
  {EndFor}
  For I := 1 To 3 Do
    For J := 1 To 3 Do
      Result [I,J] := Temp [I,J]
    {EndFor}
  {EndFor}
End; {Mat Mul}
Procedure Identity (Var Matrix: Matrix3x3);
Var
  Row: Integer;
  Column: Integer;
Begin {Identity}
  For Row := 1 To 3 Do
    For Column := 1 To 3 Do
      If Row = Column
      Then
        Matrix [Row,Column] := 1.0
      Else
        Matrix [Row,Column] := 0.0
      {EndIf}
    {EndFor}
  {EndFor}
End; {Identity}
Procedure CalcTrans;
Var
  Temp: Matrix3x3;
Begin {CalcTrans}
  Mat Mul (Temp, W To VP, VP To VD);
  Mat Mul (W To PD, Temp, VD To PD);
End; {CalcTrans}
Procedure SetWindow ((XMin, XMax, YMin, YMax: Integer));
Begin {SetWindow}
  Window [Left] := XMin;
  Window [Right] := XMax;
  Window [Bottom] := YMin;
  Window [Top] := YMax;
  Identity (W To VP);
  W To VP [1,1] := 1.0 / (XMax - XMin);
  W To VP [2,2] := 1.0 / (YMax - YMin);
  W To VP [3,1] := -XMin * W To VP [1,1];
  W To VP [3,2] := -YMin * W To VP [2,2];
  CalcTrans
End; {SetWindow}
Procedure SetViewport ((XMin, XMax, YMin, YMax: Real));
Begin {SetViewport}
  Identity (VP To VD);
  VP To VD [1,1] := XMax - XMin;
  VP To VD [2,2] := YMax - YMin;
  VP To VD [3,1] := XMin;
  VP To VD [3,2] := YMin;
  CalcTrans
End; {SetViewport}
Procedure SetDevice ((Var XMin, XMax, YMin, YMax: Integer));
Begin {SetDevice}
  Case CurrentDevice Of

```

Appendix A-3 - IoConcept Unit

```

{SR-
{SS Concept}
Unit IoConcept;
Interface
  Uses
    {SU ccutil/GraphLib.obj}
    Graphdefs;
  Function Init_Concept (Colour: ScreenColour;
    Style: LineType): Boolean;
  Procedure Inq_Concept (Var XMin, XMax, YMin, YMax,
    XDotsPerInch, YDotsPerInch: Integer);
  Procedure Set_Concept (Var XMin, XMax, YMin, YMax: Integer);
  Procedure Colour_Concept (Colour: ScreenColour);
  Procedure Style_Concept (Style: LineType);
  Procedure Point_Concept (X1, Y1: Integer);
  Procedure Line_Concept (X1, Y1, X2, Y2: Integer);
  Procedure Text_Concept (X, Y: Integer; CChars: String255;
    Theta, Slant: Real;
    Width, Height: Integer);

  Procedure Page_Concept;
  Procedure Term_Concept;
{EndInterface}
Implementation
  Const
    X Dots Inch = 72; {X dots per inch.}
    Y Dots Inch = 72; {Y dots per inch.}
    Asc_ESC = $1B;
  Type
    Byte = -128..127;
    GraphBuffer = Record
      Case Integer Of
        0: (pi: Array [1..10] Of Integer);
        1: (pb: Array [1..20] Of Byte);
      {EndCase}
    End;
  Var
    DcmIDrv: LongInt;
    Buffer: GraphBuffer;
    CurrentColour: ScreenColour;
    CurrentStyle: LineType;
  Procedure SegUnloc; External;
  Procedure SegLock; External;
  Function OSdispDv: Integer; External;
  Function Init_Concept ((Colour: ScreenColour;
    Style: LineType): Boolean);
  Begin {Init Concept}
    SegLock;
    Colour_Concept (Colour);
    Style_Concept (Style);
    Init_Concept := True;
    DcmIDrv := OSdispDv;
    Buffer.pb [1] := Asc ESC;
    Buffer.pb [2] := Ord('b');
    Buffer.pb [3] := Asc ESC;
    Buffer.pb [4] := Ord('g');
    Buffer.pb [5] := Asc ESC;
    Buffer.pb [6] := Ord('o');
    Buffer.pi [4] := 0;
    Buffer.pi [5] := 0;
    Buffer.pb [11] := 2;
    {Turn off cursor and set graphics origin.}
    UnitWrite (DcmIDrv, Buffer, 11)
  End; {Init Concept}
  Procedure Inq_Concept ((Var XMin, XMax, YMin, YMax,

```

```

Then
  Visible := False
Else
  Begin
    If C1 = 1
    Then
      C := C2
    Else
      C := C1
    (EndIf)
    If Left In C
    Then
      Begin
        Y := Y1 + (Y2 - Y1) *
          (Window [Left] - X1) Div (X2 - X1);
        X := Window [Left]
      End
    Else
      If Right In C
      Then
        Begin
          Y := Y1 + (Y2 - Y1) *
            (Window [Right] - X1) Div (X2 - X1);
          X := Window [Right]
        End
      Else
        If Bottom In C
        Then
          Begin
            X := X1 + (X2 - X1) *
              (Window [Bottom] - Y1) Div
              (Y2 - Y1);
            Y := Window [Bottom]
          End
        Else
          If Top In C
          Then
            Begin
              X := X1 + (X2 - X1) *
                (Window [Top] - Y1) Div
                (Y2 - Y1);
              Y := Window [Top]
            End
          (EndIf)
        (EndIf)
      (EndIf)
    (EndIf)
    If C = C1
    Then
      Begin
        X1 := X;
        Y1 := Y;
        Code (X1, Y1, C1)
      End
    Else
      Begin
        X2 := X;
        Y2 := Y;
        Code (X2, Y2, C2)
      End
    (EndIf)
  End
(EndIf)
(EndWhile)
Coords 2 Visible := Visible
End (Coords 2 Visible)
Function Clip 2 Coords (X1, Y1, X2, Y2: Integer;
  Var A1, B1, A2, B2: Integer): Boolean;
  Var
    X1_Temp: Integer;

```

```

Y_1_Temp:      Integer;
X_2_Temp:      Integer;
Y_2_Temp:      Integer;
Begin {Clip 2 Coords}
  X_1_Temp := X1;
  Y_1_Temp := Y1;
  X_2_Temp := X2;
  Y_2_Temp := Y2;
  If Coords_2_Visible (X_1_Temp, Y_1_Temp, X_2_Temp, Y_2_Temp)
  Then
    Begin
      Trans_2_Coords (X_1_Temp, Y_1_Temp,
                      X_2_Temp, Y_2_Temp, A1, B1, A2, B2);
      Clip_2_Coords := True
    End
  Else
    Begin
      A1 := 0;
      B1 := 0;
      A2 := 0;
      B2 := 0;
      Clip_2_Coords := False
    End
  {EndIf}
End; {Clip 2 Coords}
Function Clip_1_Coord (X, Y: Integer; Var A, B: Integer): Boolean;
Var
  Visible: Boolean;
Begin {Clip 1 Coord}
  Visible := False;
  If (X >= Window [Left]) And (X <= Window [Right])
  Then
    If (Y >= Window [Bottom]) And (Y <= Window [Top])
    Then
      Visible := True;
    {EndIf}
  {EndIf}
  If Visible
  Then
    Trans_1_Coord (X, Y, A, B)
  Else
    Begin
      A := 0;
      B := 0
    End
  {EndIf}
  Clip_1_Coord := Visible
End; {Clip 1 Coord}
Procedure Draw_Line (X_1, Y_1, X_2, Y_2: Integer);
Begin {Draw_Line}
  Case CurFentDevice Of
    Concept:
      Line_Concept (X_1, Y_1, X_2, Y_2);
    HP7221:
      Line_HP7221 (X_1, Y_1, X_2, Y_2);
    LA120:
      Line_LA120 (X_1, Y_1, X_2, Y_2)
  End
{EndCase}
End; {Draw_Line}
Procedure Plot_Point (X_1, Y_1: Integer);
Begin {Plot_Point}
  Case CurFentDevice Of
    Concept:
      Point_Concept (X_1, Y_1);
    HP7221:
      Point_HP7221 (X_1, Y_1);
    LA120:
      Point_LA120 (X_1, Y_1)
  End
End

```

```

    End; {Plot-Point}
    Procedure SetColour ((Colour: ScreenColour));
    Begin {SetColour}
        CurrentColour := Colour;
        Case CurrentDevice Of
            Concept:
                Colour_Concept (Colour);
            HP7221:
                Colour_HP7221 (Colour);
            LA120:
                Colour_LA120 (Colour);
        End
    End; {SetColour}
    Procedure SetLine ((Style: LineType));
    Begin {SetLine}
        CurrentStyle := Style;
        Case CurrentDevice Of
            Concept:
                Style_Concept (Style);
            HP7221:
                Style_HP7221 (Style);
            LA120:
                Style_LA120 (Style);
        End
    End; {SetLine}
    Procedure InqColour ((Var Colour: ScreenColour));
    Begin {InqColour}
        Colour := CurrentColour;
    End; {InqColour}
    Procedure InqStyle ((Var Style: LineType));
    Begin {InqStyle}
        Style := CurrentStyle;
    End; {InqStyle}
    Procedure SlewAbs ((X, Y: Integer));
    Begin {SlewAbs}
        RelX := 0;
        RelY := 0;
        CurrentX := X;
        CurrentY := Y;
    End; {SlewAbs}
    Procedure LineAbs ((X_1, X_2, Y_1, Y_2));
    Var
        A1: Integer;
        B1: Integer;
        A2: Integer;
        B2: Integer;
    Begin {LineAbs}
        If Clip_2_Coords (X_1, Y_1, X_2, Y_2, A1, B1, A2, B2)
        Then
            Draw Line (A1, B1, A2, B2);
        {EndIf}
        SlewAbs (X_2, Y_2)
    End; {LineAbs}
    Procedure DrawAbs ((X, Y: Integer));
    Begin {DrawAbs}
        LineAbs (CurrentX, CurrentY, X, Y)
    End; {DrawAbs}
    Procedure LineRel ((X_1, Y_1, X_2, Y_2: Integer));
    Var
        A1: Integer;
        B1: Integer;
        A2: Integer;
        B2: Integer;
    Begin {LineRel}
        If Clip_2_Coords (CurrentX + X_1,
                        CurrentY + Y_1,
                        CurrentX + X_2,

```



```

        CurrentY + Y_2, A1, B1, A2, B2)
    Then
        Draw_Line (A1, B1, A2, B2);
    {EndIf}
    SlewRel (X_2, Y_2)
End; {LineRel}
Procedure SlewRel ((X, Y: Integer));
Begin {SlewRel}
    RelX := X;
    RelY := Y;
End; {SlewRel}
Procedure DrawRel ((X, Y: Integer));
Begin {DrawRel}
    LineRel (RelX, RelY, X, Y);
End; {DrawRel}
Procedure PointRel ((X, Y: Integer));
Var
    A: Integer;
    B: Integer;
Begin {RelPoint}
    If Clip_1_Coord (CurrentX + X, CurrentY + Y, A, B)
    Then
        Plot_Point (A, B);
    {EndIf}
    SlewRel (X, Y)
End;
Procedure PointAbs ((X, Y: Integer));
Var
    A: Integer;
    B: Integer;
Begin
    If Clip_1_Coord (X, Y, A, B)
    Then
        Plot_Point (A, B);
    {EndIf}
    SlewAbs (X, Y)
End; {PointAbs}
Procedure Text_Soft (Var X, Y: Integer; Message: String255;
    Theta, Slant: Real; Width, Height: Integer);
Var
    I: Integer;
    Loc_X: Type_Coord;
    Loc_Y: Type_Coord;
    Ptr_Start: Integer;
    Ptr_End: Integer;
    X2: Integer;
    Y2: Integer;
    Radius: Real;
Begin
    Radius := Length (Message) * Width;
    X2 := X + Round (Radius * Cos (Theta));
    Y2 := Y + Round (Radius * Sin (Theta));
    For I := 1 To Length (Message) Do
        Begin
            SlewAbs (X, Y);
            Ptr_Start := Pos (Copy (Message, I, 1), Letters);
            Ptr_End := Ptrs [Ptr_Start + 1];
            Ptr_Start := Ptrs [Ptr_Start];
            While Ptr_Start < Ptr_End Do
                Begin
                    Loc_X := Coords [Ptr_Start, 1];
                    Loc_Y := Coords [Ptr_Start, 2];
                    If Loc_X > Char_Width
                    Then
                        SlewRel ((Loc_X - Char_Spacing) *
                            Width Div Char_Spacing,
                            Loc_Y * Height Div Char_Height)
                    Else
                        DrawRel (Loc_X * Width Div Char_Spacing,
                            Loc_Y * Height Div Char_Height);
                End
            End
        End
    End

```

```

        {EndIf}
        Ptr_Start := Ptr_Start + 1
    End;
    {EndWhile}
    X := X + Width
End;
{EndFor}
X := X2;
Y := Y2
End;
Procedure Text_Hard (Var X, Y: Integer; Chars: String255;
                    Theta, Slant: Real; Width, Height: Integer);
Var
    A1: Integer;
    B1: Integer;
    A2: Integer;
    B2: Integer;
    X2: Integer;
    Y2: Integer;
    Radius: Real;
Begin {DrawText}
    Radius := Length (Chars) * Width;
    X2 := Round (Radius * Cos (Theta));
    Y2 := Round (Radius * Sin (Theta));
    If Coords_2_Visible (X, Y, X2, Y2)
    Then
        Begin
            Trans_1_Coord (X, Y, A1, B1);
            Trans_1_Coord (Width, Height, A2, B2);
            Case CurrentDevice Of
                Concept:
                    Text_Concept (A1, B1, Chars, Theta,
                                   Slant, A2, B2);
                HP7221:
                    Text_HP7221 (A1, B1, Chars, Theta,
                                   Slant, A2, B2);
                LA120:
                    Text_LA120 (A1, B1, Chars, Theta,
                                   Slant, A2, B2)
            End
        {EndCase}
    End;
    {EndIf}
    X := X2;
    Y := Y2
End; {TextAbs}
Procedure DrawText (Var X, Y: Integer; Chars: String255;
                   Theta, Slant: Real; Width, Height: Integer);
Begin
    Case CurrentDevice Of
        Concept:
            Text_Soft (X, Y, Chars, Theta, Slant, Width, Height);
        HP7221:
            Text_Hard (X, Y, Chars, Theta, Slant, Width, Height);
        LA120:
            Text_Soft (X, Y, Chars, Theta, Slant, Width, Height);
    End
    {EndCase}
End;
Procedure TextRel ((X, Y: Integer; Chars: String255;
                  Theta, Slant: Real; Width, Height: Integer));
Var
    LocalX: Integer;
    LocalY: Integer;
Begin
    LocalX := CurrentX + RelX + X;
    LocalY := CurrentY + RelY + Y;
    DrawText (LocalX, LocalY, Chars,
              Theta, Slant, Width, Height);
End;

```

```

Procedure TextAbs ((X, Y: Integer; Chars: String255;
                  Theta, Slant: Real; Width, Height: Integer));
  Var
    TempX:      Integer;
    TempY:      Integer;
  Begin
    TempX := X;
    TempY := Y;
    DrawText (TempX, TempY, Chars, Theta, Slant, Width, Height);
    SlewAbs (CurrentX + TempX, CurrentY + TempY)
  End;
Procedure InqDevice ((Var XMin, XMax, YMin, YMax,
                      XDotsPerInch, YDotsPerInch: Integer));
  Begin {InqDevice}
    Case CurrentDevice Of
      Concept:
        Inq_Concept (XMin, XMax, YMin, YMax,
                     XDotsPerInch, YDotsPerInch);
      HP7221:
        Inq_HP7221 (XMin, XMax, YMin, YMax,
                    XDotsPerInch, YDotsPerInch);
      LA120:
        Inq_LA120 (XMin, XMax, YMin, YMax,
                   XDotsPerInch, YDotsPerInch);
      Otherwise;;
    End
  End; {InqDevice}
Function InitDevice (Device: DeviceType): Boolean;
  Var
    Result: Boolean;
  Begin {Init Device}
    CurrentStyle := Solid;
    Case Device Of
      Concept:
        Begin
          CurrentColour := White;
          Result := Init_Concept (CurrentColour,
                                  CurrentStyle);
          Init_Soft
        End;
      HP7221:
        Begin
          CurrentColour := Black;
          Result := Init_HP7221 (CurrentColour,
                                 CurrentStyle);
        End;
      LA120:
        Begin
          CurrentColour := Black;
          Result := Init_LA120 (CurrentColour,
                                CurrentStyle);
          Init_Soft
        End;
    End;
  {EndCase}
  If Result
  Then
    CurrentDevice := Device
  Else
    CurrentDevice := NoDevice;
  {EndIf}
  InitDevice := Result
End; {Init Device}
Procedure NewPage;
  Begin {NewPage}
    Case CurrentDevice Of
      Concept:
        Page_Concept;
      HP7221:
        Page_HP7221;
    End
  End

```

```

    LA120:
    Page_LA120;
  End
{EndCase}
End; {NewPage}
Function InitGraphics ((Device: DeviceType): Boolean);
Var
  X_Min_Port:      Integer;
  X_Max_Port:      Integer;
  Y_Min_Port:      Integer;
  Y_Max_Port:      Integer;
  XDotsPerInch:    Integer;
  YDotsPerInch:    Integer;
Begin {InitGraphics}
  If InitDevice (Device)
  Then
    Begin
      SegLock;
      SlewAbs (0, 0);
      Identity (W To VP);
      Identity (VP To VD);
      Identity (VD To PD);
      SetWindow (0, 1, 0, 1);
      InqDevice (X_Min_Port, X_Max_Port, Y_Min_Port,
        Y_Max_Port, XDotsPerInch, YDotsPerInch);
      SetDevice (X_Min_Port, X_Max_Port,
        Y_Min_Port, Y_Max_Port);
      InitGraphics := True
    End
  Else
    InitGraphics := False
  {EndIf}
End; {InitGraphics}
Procedure TermGraphics;
Begin {TermGraphics}
  Case CurrentDevice Of
    Concept:
      Term_Concept;
      HP7221:
        Term_HP7221;
      LA120:
        Term_LA120
    End;
  {EndCase}
  SegUnloc
End; {TermGraphics}
{$S}
{EndImplementation}
End. {Graph}

```

Appendix A-3 - IoConcept Unit

```

SR-
{SS Concept}
Unit IoConcept;
Interface
  Uses
    {SU ccutil/GraphLib.obj}
    Graphdefs;
  Function Init_Concept (Colour: ScreenColour;
    Style: LineType): Boolean;
  Procedure Inq_Concept (Var XMin, XMax, YMin, YMax,
    XDotsPerInch, YDotsPerInch: Integer);
  Procedure Set_Concept (Var XMin, XMax, YMin, YMax: Integer);
  Procedure Colour_Concept (Colour: ScreenColour);
  Procedure Style_Concept (Style: LineType);
  Procedure Point_Concept (X 1, Y 1: Integer);
  Procedure Line_Concept (X 1, Y 1, X 2, Y 2: Integer);
  Procedure Text_Concept (X, Y: Integer; Chars: String255;
    Theta, Slant: Real;
    Width, Height: Integer);

  Procedure Page_Concept;
  Procedure Term_Concept;
{EndInterface}
Implementation
  Const
    X Dots Inch = 72; {X dots per inch.}
    Y Dots Inch = 72; {Y dots per inch.}
    AEsc_ESC = $1B;
  Type
    Byte = -128..127;
    GraphBuffer = Record
      Case Integer Of
        0: (pi: Array [1..10] Of Integer);
        1: (pb: Array [1..20] Of Byte);
      {EndCase}
    End;
  Var
    DcmIDrv: LongInt;
    Buffer: GraphBuffer;
    CurrentColour: ScreenColour;
    CurrentStyle: LineType;
  Procedure SegUnloc; External;
  Procedure SegLock; External;
  Function OSdispDv: Integer; External;
  Function Init_Concept ((Colour: ScreenColour;
    Style: LineType): Boolean);
  Begin {Init_Concept}
    SegLock;
    Colour_Concept (Colour);
    Style_Concept (Style);
    Init_Concept := True;
    DcmIDrv := OSdispDv;
    Buffer.pb [1] := Asc ESC;
    Buffer.pb [2] := Ord('^b');
    Buffer.pb [3] := Asc ESC;
    Buffer.pb [4] := Ord('^g');
    Buffer.pb [5] := Asc ESC;
    Buffer.pb [6] := Ord('^o');
    Buffer.pi [4] := 0;
    Buffer.pi [5] := 0;
    Buffer.pb [11] := 2;
    {Turn off cursor and set graphics origin.}
    UnitWrite (DcmIDrv, Buffer, 11)
  End; {Init_Concept}
  Procedure Inq_Concept ((Var XMin, XMax, YMin, YMax,

```

```

                                XDotsPerInch, YDotsPerInch: Integer));
Var
  DispRec:      Record
                  X Home:      Integer;
                  Y Home:      Integer;
                  X Size:      Integer;
                  Y Size:      Integer;
                End;
Begin {Inq Concept}
  UnitStatus (Dcm1Drv, DispRec, 5);
  XMin := 0;
  XMax := DispRec.X_Size - 1;
  YMin := 0;
  YMax := DispRec.Y_Size - 1;
  XDotsPerInch := X_Dots_Inch;
  YDotsPerInch := Y_Dots_Inch;
End; {Inq Concept}
Procedure Set_Concept ((Var XMin, XMax, YMin, YMax: Integer));
Var
  X Dots:      Integer;
  Y Dots:      Integer;
Begin {Set Concept}
  Inq_Concept (XMin, XMax, YMin, YMax, X_Dots, Y_Dots)
End; {Set Concept}
Procedure Colour_Concept ((Colour: ScreenColour));
Begin {Colour_Concept}
  CurrentColour := Colour
End; {Colour_Concept}
Procedure Style_Concept ((Style: LineType));
Begin {Style_Concept}
  CurrentStyle := Style
End; {Style_Concept}
Procedure Point_Concept ((X_1, Y_1: Integer));
Begin {Point_Concept}
  Buffer.pb [1] := Asc ESC;
  Buffer.pb [2] := Ord('p');
  Buffer.pl [2] := X_1;
  Buffer.pl [3] := Y_1;
  Case CurrentColour Of
    Black:
      Buffer.pb [7] := 0 Mod 256;
    Flip:
      Buffer.pb [7] := -1 Mod 256;
    Otherwise:
      Buffer.pb [7] := 1 Mod 256;
  End;
End; {Point_Concept}
UnitWrite (Dcm1Drv, Buffer, 7)
End; {Point_Concept}
Procedure Line_Concept ((X_1, Y_1, X_2, Y_2: Integer));
Begin {Line_Concept}
  If (CurrentStyle <> Dotted) Or
    ((X_1 <> X_2) And (Y_1 <> Y_2))
  Then
    Begin
      Buffer.pb [1] := Asc ESC;
      Buffer.pb [2] := Ord('l');
      Buffer.pl [2] := X_1;
      Buffer.pl [3] := Y_1;
      Buffer.pl [4] := X_2;
      Buffer.pl [5] := Y_2;
      Case CurrentColour Of
        Black:
          Buffer.pb [11] := 0 Mod 256;
        Flip:
          Buffer.pb [11] := -1 Mod 256;
        Otherwise:
          Buffer.pb [11] := 1 Mod 256;
      End
    End;
  End; {Line_Concept}
End; {Line_Concept}

```

```

End
Else
  If X_1 = X_2
  Then
    Begin
      Buffer.pb [1] := Asc ESC;
      Buffer.pb [2] := Ord('^f');
      Buffer.pi [2] := X_1;
      Buffer.pi [3] := Min(Y_1, Y_2);
      Buffer.pi [4] := Max(Y_1, Y_2) -
        Min(Y_1, Y_2) + 1;
      Buffer.pi [5] := 1;
      Buffer.pb [11] := 2
    End
  Else
    Begin
      Buffer.pb [1] := Asc ESC;
      Buffer.pb [2] := Ord('^f');
      Buffer.pi [2] := Min(X_1, X_2);
      Buffer.pi [3] := Y_1;
      Buffer.pi [4] := 1;
      Buffer.pi [5] := Max(X_1, X_2) -
        Min(X_1, X_2) + 1;
      Buffer.pb [11] := 4
    End
  End;
EndIf
EndIf
UnitWrite (DcmlDrv, Buffer, 11)
End; {Line Concept}
Procedure Text Concept;
Begin {Text Concept}
End; {Text Concept}
Procedure Page Concept;
Begin {Page Concept}
  Buffer.pb [1] := Asc ESC;
  Buffer.pb [2] := Ord('^J');
  UnitWrite (DcmlDrv, Buffer, 2)
End; {Page Concept}
Procedure Term Concept;
Begin {Term Concept}
  Buffer.pb [1] := Asc ESC;
  Buffer.pb [2] := Ord('^c');
  Buffer.pb [3] := Asc ESC;
  Buffer.pb [4] := Ord('^t');
  UnitWrite (DcmlDrv, Buffer, 4); {Turn on cursor.}
  SegUnloc
End; {Term Concept}
EndImplementation
{$$}
End. {Concept}

```

Appendix A-4 - IoLA120 Unit

```

{SR-}
{SS LA120}
Unit IoLA120;
Interface
  Uses
    {SU /ctutil/GraphLib.Obj}
  GraphDefs;
Function Init_LA120 (Colour: ScreenColour;
  Style: LineType): Boolean;
Procedure Inq_LA120 (Var XMin, XMax, YMin, YMax,
  XDotsPerInch, YDotsPerInch: Integer);
Procedure Set_LA120 (Var XMin, XMax, YMin, YMax: Integer);
Procedure Colour_LA120 (Colour: ScreenColour);
Procedure Style_LA120 (Style: LineType);
Procedure Point_LA120 (Var X 1, Y 1: Integer);
Procedure Line_LA120 (Var X 1, Y 1, X 2, Y 2: Integer);
Procedure Text_LA120 (X, Y: Integer; Chars: String255;
  Theta, Slant: Real;
  Width, Height: Integer);

Procedure Page_LA120;
Procedure Term_LA120;
{EndInterface}
Implementation
Const
  X Dots Per Inch = 110;
  Y Dots Per Inch = 72;
  SelanaF = 1451;
  PlotFile = 'LA120.PLOT';
Type
  Type_Bits = Packed Array [0..7] Of Boolean;
  Type_Byte = -128..127;
  Type_Ghost = Record
    Case Boolean Of
      True: (Bits: Type_Bits);
      False: (Byte: Type_Byte);
    End;
  {EndRecord}
  Type_DDA = Record
    Case Boolean Of
      True: (Bits32: LongInt);
      False: (Whole: Integer;
        Frac: Integer);
    End;
  {EndRecord}
  pType_Line = Type_Line;
  Type_Line = Record
    Link: pType_Line;
    X Current: Type_DDA;
    Y Current: Type_DDA;
    X Increment: LongInt;
    Y Increment: LongInt;
    Iterations: Integer;
  End;
  pType_Point = Type_Point;
  Type_Point = Record
    Link: pType_Point;
    X Current: Integer;
    Y Current: Integer;
  End;
Var
  CurrentColour: ScreenColour;
  CurrentStyle: LineType;
  Two To The N: Array [0..14] Of Integer;
  Head Lines: pType_Line;
  Head Points: pType_Point;
  Start Lines: pType_Line;

```



```

Start Points:      pType Point;
Heap Base:         pType Line;
Page Height:       Integer;
Display:           Interactive;
Buffer:            Array [0..Selanar] Of Type_Ghost;
Dirty Buffer:      Boolean;
Procedure SegUnloc; External;
Procedure SegLock; External;
Procedure Set LA120 ((Var XMin, XMax, YMin, YMax: Integer));
Begin {Set LA120}
  XMax := Min (Selanar, Max (XMax, 0));
  YMax := Min (Selanar, Max (YMax, 0));
  Page Height := YMax
End; {Set LA120}
Procedure Inq LA120 ((Var XMin, XMax, YMin, YMax,
  XDotsPerInch, YDotsPerInch: Integer));
Begin {Inq LA120}
  XMin := 0;
  XMax := Selanar;
  YMin := 0;
  YMax := Page Height;
  XDotsPerInch := X Dots Per Inch;
  YDotsPerInch := Y Dots Per Inch;
End; {Inq LA120}
Procedure Emit_Char (Argument: Type_Byte; Count: Integer);
Const
  Offset := 63;
Begin {Emit Char}
  If Count <> 1
  Then
    Write (Display, '!', Count: 1);
  {EndIf}
  Write (Display, Chr (Argument + Offset))
End; {Emit Char}
Procedure New_Line;
Begin {New_Line}
  Write (Display, '-')
End; {New_Line}
Procedure Reset_Buffer;
Var
  Column: Integer;
Begin {Reset Buffer}
  For Column := 0 To Selanar Do
    Buffer[Column].Byte := 0;
  {EndFor}
  Dirty Buffer := False
End; {Reset Buffer}
Procedure Dump_Buffer (Current_Line: Integer);
Const
  Blank = 0;
Var
  Ch: Type_Byte;
  Ptr: Integer;
  Count: Integer;
  Column: Integer;
Begin {Dump Buffer}
  If Current_Line Mod 6 = 0
  Then
    Begin
      If Dirty_Buffer
      Then
        Begin
          Ch := Buffer[0].Byte;
          Count := 1;
          Ptr := 1;
          Repeat
            If Buffer[Ptr].Byte = Ch
            Then
              Count := Count + 1
            Else

```

```

        Begin
            Emit_Char (Ch, Count);
            Ch := Buffer[Ptr].Byte;
            Count := 1
        End;
        {EndIf}
        Ptr := Ptr + 1
    Until Ptr > Selanar;
    If Ch <> Blank
    Then
        Emit_Char (Ch, Count);
    {EndIf}
    Reset_Buffer
    End;
    {EndIf}
    New_Line
End
{EndIf}
End; {Dump Buffer}
Procedure Plot_Point (Column, Row: Integer);
Begin
    Buffer[Column].Bits[5 - Row Mod 6] := True;
    Dirty_Buffer := True
End;
Procedure Insert_Point (Var Object: pType_Point);
Var
    Spy:                pType_Point;
    Temp:                pType_Point;
    Searching:           Boolean;
Begin {Insert Point}
    Spy := Head_Points;
    Temp := Head_Points;
    Searching := True;
    While Searching Do
        If Spy = Nil
        Then
            Searching := False
        Else
            If Spy^.Y_Current > Object^.Y_Current
            Then
                Begin
                    Temp := Spy;
                    Spy := Spy.Link
                End
            Else
                Searching := False;
            {EndIf}
        {EndIf}
    {EndWhile}
    Object^.Link := Spy;
    If Spy = Head_Points
    Then
        Head_Points := Object
    Else
        Temp^.Link := Object;
    {EndIf}
    Object := Nil
End; {Insert Point}
Procedure Insert_Line (Var Object: pType_Line);
Var
    Spy:                pType_Line;
    Temp:                pType_Line;
    Searching:           Boolean;
Begin {Insert Line}
    Spy := Head_Lines;
    Temp := Head_Lines;
    Searching := True;
    While Searching Do
        If Spy = Nil
        Then

```

2

```

        Searching := False
    Else
        If Spy^.Y_Current.Whole > Object^.Y_Current.Whole
        Then
            Begin
                Temp := Spy;
                Spy := Spy^.Link
            End
        Else
            Searching := False;
        {EndIf}
    {EndIf}
{EndWhile}
Object^.Link := Spy;
If Spy = Head_Lines
Then
    Head_Lines := Object
Else
    Temp^.Link := Object;
{EndIf}
Object := Nil
End; {Insert Line}
Procedure Page_LA120;
Begin {Page_LA120}
    Page (Display)
End; {Page_LA120}
Procedure Point_LA120 ((Var X_1, Y_1: Integer));
Var
    Object:      pType_Point;
Begin {Point LA120}
    New (Object);
    With Object^ Do
        Begin
            Y_Current := Y_1;
            X_Current := X_1;
        End;
    {EndWith}
    Insert Point (Object);
End; {Point LA120}
Procedure Line_LA120 ((Var X_1, Y_1, X_2, Y_2: Integer));
Var
    Delta_X:      Integer;
    Delta_Y:      Integer;
    N:            Integer;
    Biggest:      Integer;
    Object:      pType_Line;
Begin {Line LA120}
    New (Object);
    Delta_Y := Y_1 - Y_2;
    Delta_X := X_1 - X_2;
    Biggest := Max (Abs (Delta_X), Abs (Delta_Y));
    N := 1;
    While Biggest >= Two_To_The_N [N] Do
        N := N + 1;
    {EndWhile}
    With Object^ Do
        Begin
            Iterations := Two_To_The_N [N] + 1;
            Y_Current.Frac := -32768;
            X_Current.Frac := -32768;
            If Y_1 < Y_2
            Then
                Begin
                    Y_Current.Whole := Y_2;
                    X_Current.Whole := X_2;
                    X_Increment := Delta_X *
                        Ord4 (Two_To_The_N [16 - N]);
                    Y_Increment := Delta_Y *
                        Ord4 (Two_To_The_N [16 - N])
                End
            End
        End
    End
End

```

```

Else
  Begin
    Y_Current.Whole := Y_1;
    X_Current.Whole := X_1;
    X_Increment := - Delta X *
      Ord4 (Two_To_The_N [16 - N]);
    Y_Increment := - Delta Y *
      Ord4 (Two_To_The_N [16 - N]);
  End;
{EndIf}
End;
{EndWith}
Insert_Line (Object);
End; {Line LA120}
Procedure PlotLines (Active_Line: Integer);
Var
  Looking: Boolean;
  Active_Object: pType_Line;
  End_Lines: pType_Line;
Begin {PlotLines}
  If Start_Lines <> Nil
  Then
    If Start_Lines^.Y_Current.Whole = Active_Line
    Then
      Begin
        *****
        *
        * Isolate the active vectors for this scan line.
        *
        *
        * Looking := True;
        * End_Lines := Start_Lines;
        * While Looking Do
        *   If End_Lines = Nil
        *   Then
        *     Looking := False
        *   Else
        *     If (Start_Lines^.Y_Current.Whole =
        *         End_Lines^.Y_Current.Whole) Or
        *         (End_Lines^.Iterations <= 0)
        *     Then
        *       End_Lines := End_Lines^.Link
        *     Else
        *       Looking := False;
        *     {EndIf}
        *   {EndIf}
        * {EndWhile}
        * Active_Object := Start_Lines;
        *
        * *****
        *
        * Plot all the active vectors for this scan line.
        *
        *
        * While Active_Object <> End_Lines Do
        *   Begin
        *     With Active_Object Do
        *       While (Y_Current.Whole = Active_Line)
        *         And (Iterations > 0) Do
        *           Begin
        *             Plot_Point (X_Current.Whole,
        *                         Y_Current.Whole);
        *             X_Current.Bits32 :=
        *               - X_Current.Bits32 +
        *               X_Increment;
        *             Y_Current.Bits32 :=
        *               - Y_Current.Bits32 +
        *               Y_Increment;
        *             Iterations := Iterations - 1
        *           End;
        *         End;
      End;
    End;
  End;
End;

```

```

      {EndWhile}
    {EndWith}
    Active_Object := Active_Object^.Link
  End;
{EndWhile}
*****
Move head pointer forward over the inactive vectors.
-----

    Looking := True;
    While Looking Do
      If Start_Lines = Nil
      Then
        Looking := False
      Else
        If Start_Lines^.Iterations <= 0
        Then
          Start_Lines := Start_Lines^.Link
        Else
          Looking := False
        {EndIf}
      {EndIf}
    {EndWhile}
  *****
End
{EndIf}
{EndIf}
End; {PlotLines}
Procedure PlotPoints (Active_Line: Integer);
Var
  Looking: Boolean;
  End Points: pType_Point;
Begin {PlotPoints}
  If Start_Points <> Nil
  Then
    If Start_Points^.Y_Current = Active_Line
    Then
      Begin
        *****
        Isolate the active points for this scan line.
        -----

          End Points := Start_Points;
          Looking := True;
          While Looking Do
            If End_Points = Nil
            Then
              Looking := False
            Else
              If End_Points^.Y_Current =
                Start_Points^.Y_Current
              Then
                End_Points := End_Points^.Link
              Else
                Looking := False;
              {EndIf}
            {EndIf}
          {EndWhile}
        *****
        Plot all the active points for this scan line.
        -----

          While Start_Points <> End_Points Do
            With Start_Points Do

```

```

Begin
  Plot Point (X Current, Y Current);
  Start_Points := Start_Points + 1;
End
{EndWith}
{EndWhile}
*****

End
{EndIf}
{EndIf}
End; {Plot Points}
Procedure Plot;
Var
  Current Line: Integer;
Begin {Plot}
  Start Lines := Head Lines;
  Start_Points := Head Points;
  Current Line := Page Height;
  While Current Line >= 0 Do
    Begin
      PlotLines (Current Line);
      PlotPoints (Current Line);
      Dump Buffer (Current Line);
      Current Line := Current Line - 1;
    End;
  {EndWhile}
End; {Plot}
Procedure Colour_LAL20 ((Colour: ScreenColour));
Begin {Colour_LAL20}
  CurrentColour := Colour
End; {Colour_LAL20}
Procedure Style_LAL20 ((Style: LineType));
Begin {Style_LAL20}
  CurrentStyle := Style
End; {Style_LAL20}
Procedure Text_LAL20 ((X, Y: Integer; Chars: String255;
  Theta, Slant: Real;
  Width, Height: Integer));
Begin {Text_LAL20}
End; {Text_LAL20}
Function Init_LAL20 ((Colour: ScreenColour;
  Style: LineType): Boolean);
Begin {Init_LAL20}
  SegLock :=
  Colour_LAL20 (Colour);
  Style_LAL20 (Style);
  Init_LAL20 := True;
  Head Lines := Nil;
  Head Points := Nil;
  Mark (Heap Base);
  Page Height := Selanar;
  Two To The N 0 := 1;
  Two To The N 1 := 2;
  Two To The N 2 := 4;
  Two To The N 3 := 8;
  Two To The N 4 := 16;
  Two To The N 5 := 32;
  Two To The N 6 := 64;
  Two To The N 7 := 128;
  Two To The N 8 := 256;
  Two To The N 9 := 512;
  Two To The N 10 := 1024;
  Two To The N 11 := 2048;
  Two To The N 12 := 4096;
  Two To The N 13 := 8192;
  Two To The N 14 := 16384;
  Rewrite (Display, PlotFile);
  Writeln (Display);
  Write (Display, Chr (27), 'POq');

```

```
      Reset Buffer;
    End; {Init LA120}
  Procedure Term LA120;
  Begin {Term LA120}
    Plot;
    Writeln (Display, Chr (27), '~');
    Close (Display, Lock);
    Release (Heap_Base);
    SegUnloc
  End; {Term LA120}
  {SS}
  {EndImplementation}
End.
```


Appendix A-5 - IOHP7221 Unit

```

{SR-}
{SS HP7221}
Unit IOHP7221;
Interface
  Uses
    {SU /ccutil/GraphLib.Obj}
    GraphDefs;
  Function Init_HP7221      (Colour: ScreenColour;
                             Style: LineType): Boolean;
  Procedure Inq_HP7221      (Var XMin, XMax, YMin, YMax,
                             XDotsPerInch, YDotsPerInch: Integer);
  Procedure Set_HP7221      (Var XMin, XMax, YMin, YMax: Integer);
  Procedure Colour_HP7221   (Colour: ScreenColour);
  Procedure Style_HP7221    (Style: LineType);
  Procedure Move_HP7221     (X, Y: Integer);
  Procedure Draw_HP7221     (X, Y: Integer);
  Procedure Point_HP7221    (X1, Y1: Integer);
  Procedure Line_HP7221     (X1, Y1, X2, Y2: Integer);
  Procedure Text_HP7221     (X, Y: Integer; Chars: String255;
                             Theta, Slant: Real;
                             Width, Height: Integer);

  Procedure Page_HP7221;
  Procedure Term_HP7221;
{EndInterface}
Implementation
  {The data encoding algorithms are found
   in: Hewlett-Packard 7221C and 7221T Graphics Plotter
   Operating and Programming Manual}
  Const
    {Default HP7221 (horizontal) X min.}
    X Min Port      = 0;
    {Default HP7221 (horizontal) X max.}
    X Max Port      = 2235;
    {HP7221 (horizontal) X units per millimeter.}
    X Dots Millimeter = 8;
    {Default HP7221 (vertical) Y min.}
    Y Min Port      = 0;
    {Default HP7221 (vertical) Y max.}
    Y Max Port      = 1727;
    {HP7221 (vertical) Y units per millimeter.}
    Y Dots Millimeter = 8;
    Buffer_Size      = 256;
  Type
    Byte            = -128..127;
    GraphBuffer     = Record
      Case Integer Of
        0: (pc: Packed Array
             [1..Buffer_Size] Of Char);
        1: (pb: Packed Array
             [1..Buffer_Size] Of Byte)
      (EndCase)
    End;
    TypeDevice      = String[8];
    pTypeDevice     = ^TypeDevice;
    String80        = String [80];
    pString80       = ^String80;
  Var
    Left:           Integer;
    Right:          Integer;
    Bottom:         Integer;
    Top:            Integer;
    IO Unit:        Integer;
    Buffer Pointer:  LongInt;
    Buffer?:         GraphBuffer;

```

```

CurrentColour:      ScreenColour;
CurrentStyle:       LineType;
Old X:              Integer;
Old Y:              Integer;
Mode:               (Mode_Line, Mode_Point, Mode_Label,
                    Mode_Colour, Mode_None);

Procedure SegUnloc; External;
Procedure SegLock; External;
Function pOSdevNam (UnitNo: Integer): pString80; External;
Function OSdcmIDv: Integer; External;
Procedure Transmit;
  Const
    Get_Size      = 0; {Returns space remaining in driver.}
  Var
    Space:         Integer;
  Begin {Transmit}
    If Buffer_Pointer = Buffer_Size
    Then
      Begin
        Repeat
          UnitStatus (IO Unit, Space, Get_Size)
        Until Space >= Buffer_Size;
        UnitWrite (IO Unit, Buffer, Buffer_Pointer);
        Buffer_Pointer := 0
      End
    {EndIf}
  End; {Transmit}
Procedure PurgeBuffer;
  Var
    I:              Integer;
  Begin {PurgeBuffer}
    For I := Buffer_Pointer + 1 To Buffer_Size Do
      Buffer.pb[I] := 0;
    {EndFor}
    Buffer_Pointer := Buffer_Size;
    Transmit
  End; {PurgeBuffer}
Procedure InsertCh (Arg: Char);
  Begin {InsertCh}
    Buffer_Pointer := Buffer_Pointer + 1;
    Buffer.pc [Buffer_Pointer] := Arg;
    Transmit
  End; {InsertCh}
Procedure Insert (Arg: Integer);
  Begin {Insert}
    Buffer_Pointer := Buffer_Pointer + 1;
    Buffer.pb [Buffer_Pointer] := Arg Mod 256;
    Transmit
  End; {Insert}
Procedure Stuff (Buf: String255);
  Var
    I:              Integer;
  Begin {Stuff}
    For I := 1 To Length (Buf) Do
      InsertCh (Buf[I])
    {EndFor}
  End; {Stuff}
Procedure S_B_N (Argument: Integer);
  Begin {S_B_N}
    If Argument <= 31
    Then
      Argument := Argument + 64;
    {EndIf}
    Insert (Argument)
  End; {S_B_N}
Procedure H_B_N (Argument: Integer);
  Var
    NN1:            Integer;
    NN2:            Integer;
    NN3:            Integer;

```

```

Procedure Asm_2_Byte;
  Var
    NP1: Integer;
    NP2: Integer;
  Begin {Asm 2 Byte}
    NP2 := NN2;
    If NP2 <= 31
    Then
      NP2 := NP2 + 64;
    {EndIf}
    NP1 := NN1 + 96;
    Insert (NP1);
    Insert (NP2);
  End; {Asm 2 Byte}
Procedure Asm_3_Byte;
  Var
    NP3: Integer;
  Begin {Asm 3 Byte}
    NP3 := NN3;
    If NP3 <= 31
    Then
      NP3 := NP3 + 64;
    {EndIf}
    Asm_2_Byte;
    Insert (NP3);
  End; {Asm 3 Byte}
Procedure One_Byte (Argument: Integer);
  Begin {One_Byte}
    Insert (Argument + 96)
  End; {One_Byte}
Procedure Two_Byte (Argument: Integer);
  Begin {Two_Byte}
    NN1 := Argument Div 64;
    NN2 := Argument Mod 64;
    Asm_2_Byte
  End; {Two_Byte}
Procedure Three_Byte (Argument: Integer);
  Var
    NR: Integer;
  Begin {Three_Byte}
    NN1 := Argument Div 4096;
    NR := Argument Mod 4096;
    NN2 := NR Div 64;
    NN3 := NR Mod 64;
    Asm_3_Byte
  End; {Three_Byte}
Begin {M B N}
  If Argument < 16
  Then
    One_Byte (Argument)
  Else
    If Argument < 1024
    Then
      Two_Byte (Argument)
    Else
      Three_Byte (Argument)
    {EndIf}
  {EndIf}
End; {M B N}
Procedure M_B_P (X, Y: Integer);
  Var
    N: Integer;
  Procedure One_Byte (X, Y: Integer);
    Begin {One_Byte}
      Insert (Y + 96 + 4 * X)
    End; {One_Byte}
  Procedure Two_Byte (X, Y: Integer);
    Var
      NX1: Integer;
      NX2: Integer;

```

```

    NP1:      Integer;
    NP2:      Integer;
  Begin {Two Byte}
    NX1 := X Div 2;
    NX2 := X Mod 2;
    NP1 := NX1 + 96;
    NP2 := Y + 32 * NX2;
    If NP2 <= 31
    Then
      NP2 := NP2 + 64;
    {EndIf}
    Insert (NP1);
    Insert (NP2);
  End; {Two Byte}
Procedure Three_Byte (X, Y: Integer);
Var
  NX1:      Integer;
  NX2:      Integer;
  NY2:      Integer;
  NY3:      Integer;
  NP1:      Integer;
  NP2:      Integer;
  NP3:      Integer;
  Begin {Three Byte}
    NX1 := X Div 16;
    NX2 := X Mod 16;
    NY2 := Y Div 64;
    NY3 := Y Mod 64;
    NP1 := NX1 + 96;
    NP2 := NY2 + 4 * NX2;
    NP3 := NY3;
    If NP2 <= 31
    Then
      NP2 := NP2 + 64;
    {EndIf}
    If NP3 <= 31
    Then
      NP3 := NP3 + 64;
    {EndIf}
    Insert (NP1);
    Insert (NP2);
    Insert (NP3);
  End; {Three Byte}
Procedure Four_Byte (X, Y: Integer);
Var
  NX1:      Integer;
  NX2:      Integer;
  NX3:      Integer;
  NY3:      Integer;
  NY4:      Integer;
  NP1:      Integer;
  NP2:      Integer;
  NP3:      Integer;
  NP4:      Integer;
  Begin {Four Byte}
    NX1 := X Div 128;
    NX2 := X Mod 128;
    NX3 := NX2 Div 2;
    NX4 := NX2 Mod 2;
    NY3 := Y Div 64;
    NY4 := Y Mod 64;
    NP1 := 96 + NX1;
    NP2 := NX2;
    If NP2 <= 31
    Then
      NP2 := NP2 + 64;
    {EndIf}
    NP3 := NY3 + 32 * NX3;
    If NP3 <= 31

```

```

Then
  NP3 := NP3 + 64;
{EndIf}
NP4 := NY4;
If NP4 <= 31
Then
  NP4 := NP4 + 64;
{EndIf}
Insert (NP1);
Insert (NP2);
Insert (NP3);
Insert (NP4);
End; {Four Byte}
Procedure Five_Byte (X, Y: Integer);
Var
  NX1: Integer;
  NX2: Integer;
  NX3: Integer;
  NY3: Integer;
  NYR: Integer;
  NY4: Integer;
  NY5: Integer;
  NP1: Integer;
  NP2: Integer;
  NP3: Integer;
  NP4: Integer;
  NP5: Integer;
Begin {Five Byte}
  NX1 := X Div 1024;
  NX2 := X Mod 1024;
  NX3 := NX2 Div 16;
  NX3 := NX2 Mod 16;
  NY3 := Y Div 4096;
  NYR := Y Mod 4096;
  NY4 := NYR Div 64;
  NY5 := NYR Mod 64;
  NP1 := NX1 + 96;
  NP2 := NX2;
  If NP2 <= 31
  Then
    NP2 := NP2 + 64;
  {EndIf}
  NP3 := NY3 + 4 * NX3;
  If NP3 <= 31
  Then
    NP3 := NP3 + 64;
  {EndIf}
  NP4 := NY4;
  If NP4 <= 31
  Then
    NP4 := NP4 + 64;
  {endIf}
  NP5 := NY5;
  If NP5 <= 31
  Then
    NP5 := NP5 + 64;
  {EndIf}
  Insert (NP1);
  Insert (NP2);
  Insert (NP3);
  Insert (NP4);
  Insert (NP5);
End; {Five Byte}
Begin {M B P}
If X > Y
Then
  N := X
Else
  N := Y;

```

```

      {EndIf}
      If N < 256
      Then
        If N > 31
        Then
          Three_Byte (X, Y)
        Else
          If N > 3
          Then
            Two_Byte (X, Y)
          Else
            One_Byte (X, Y)
          {EndIf}
        {EndIf}
      Else
        If N < 2048
        Then
          Four_Byte (X, Y)
        Else
          Five_Byte (X, Y)
        {EndIf}
      {EndIf}
    End; {M B P}
  Procedure M_B_A (Angle: Real);
  Var
    NP1: Integer;
    NP2: Integer;
    NP3: Integer;
    NA: Integer;
    NA1: Integer;
    NA2: Integer;
    NA3: Integer;
    NR: Integer;
  Begin
    If Angle >= 180.0
    Then
      Begin
        NP1 := 8;
        Angle := Angle - 180.0
      End
    Else
      NP1 := 0;
    {EndIf}
    NA := Trunc ((Angle / 90.0) * 16384);
    NA1 := NA Div 4096;
    NP1 := NP1 + NA1 + 96;
    NR := NA Mod 4096;
    Insert (NP1);
    If NR <> 0
    Then
      Begin
        NA2 := NR Div 64;
        NA3 := NR Mod 64;
        NP2 := NA2;
        If NP2 <= 31
        Then
          NP2 := NP2 + 64;
        {EndIf}
        Insert (NP2);
        If NA3 <> 0
        Then
          Begin
            NP3 := NA3;
            If NP3 <= 31
            Then
              NP3 := NP3 + 64;
            {EndIf}
            Insert (NP3)
          End
        {EndIf}
      End
    End
  End
{EndIf}

```

```

      End
    {EndIf}
  End; {M B A}
Procedure CoTour HP7221 {(Colour: ScreenColour)};
Begin {Colour HP7221}
  If {Colour <> CurrentColour} Or (Mode = Mode_None)
  Then
    Begin
      CurrentColour := Colour;
      InsertCh ('v');
      S B N (Ord (CurrentColour)); {Fix this}
      Mode := Mode_Colour
    End;
  {EndIf}
End; {Colour HP7221}
Procedure Style HP7221 {(Style: LineType)};
Begin {Style HP7221}
  If {Style <> CurrentStyle} Or (Mode = Mode_None)
  Then
    Begin
      CurrentStyle := Style;
      Case Style Of
        Dotted:
          Begin
            Stuff ('~R');
            S B N (32 + 1);
            S B N (1);
            M B N (16);
            Insertch ('~');
          End;
        Dashed:
          Begin
            Stuff ('~R');
            S B N (32 + 3);
            S B N (1);
            S B N (32 + 1);
            S B N (1);
            M B N (48);
            Insertch ('~');
          End;
        Solid:
          Stuff ('~Q');
        End;
      {EndCase}
      Mode := Mode_Colour
    End;
  {EndIf}
End; {Style HP7221}
Procedure Point HP7221;
Begin {Point HP7221}
End; {Point HP7221}
Procedure Page HP7221;
Begin {Page HP7221}
  Colour HP7221 (None);
  InsertCh ('~');
  PurgeBuffer
End; {Page HP7221}
Procedure Move HP7221 {(X, Y: Integer)};
Begin {Move HP7221}
  Insertch ('p');
  M B P (X, Y);
  Mode := Mode_Line;
  Old X := X;
  Old Y := Y;
End; {Move HP7221}
Procedure Draw HP7221 {(X, Y: Integer)};
Begin
  If Mode <> Mode_Line
  Then
    Begin

```

```

        InsertCh (~q~);
        Mode := Mode_Line
    End;
    {EndIf}
    M B P (X, Y);
    OldX := X;
    OldY := Y
End; {Draw HP7221}
Procedure Line_HP7221 ((X_1, Y_1, X_2, Y_2: Integer));
Begin {Line_HP7221}
    If (Mode <> Mode_Line) Or (X_1 <> OldX) Or (Y_1 <> OldY)
    Then
        Move_HP7221 (X_1, Y_1);
    {EndIf}
    Draw_HP7221 (X_2, Y_2)
End; {Line_HP7221}
Procedure Text_HP7221 ((X, Y: Integer; Chars: String255;
                        Theta, Slant: Real;
                        Width, Height: Integer));
Begin {Text_HP7221}
    {Move pen to appropriate location.}
    Move_HP7221 (X, Y);
    {Clear any previous rotations and set character slant.}
    Stuff (~w/);
    M B A (Slant);
    {Set string orientation angle.}
    Stuff (~w);
    M B A (Theta);
    {Set character size.}
    Stuff (~x);
    {Character Width := 3 * Width Div 2}
    M B P (Width, 2 * Height);
    {And finally draw the text.}
    Stuff (~---);
    Stuff (Chars);
    Stuff (~--);
    Mode := Mode_None;
End; {Text_HP7221}
Procedure Term_HP7221;
Begin {Term_HP7221}
    Colour_HP7221 (None);
    Stuff (~B.);
    PurgeBuffer;
    SegUnloc
End; {Term_HP7221}
Procedure Set_HP7221 ((Var XMin, XMax, YMin, YMax: Integer));
Begin {Set_HP7221}
    XMin := Max (0, Min (3200, XMin));
    YMin := Max (0, Min (2280, YMin));
    XMax := Min (3200, Max (0, XMax));
    YMax := Min (2280, Max (0, YMax));
    Stuff (~-W);
    M B P (XMin * 5, YMin * 5);
    M B P (XMax * 5, YMax * 5);
    Stuff (~-S-);
    M B P (XMax - XMin, YMax - YMin);
    InsertCh (~-);
    Left := XMin;
    Right := XMax;
    Bottom := YMin;
    Top := YMax;
End; {Set_HP7221}
Function LoadDriver: Boolean;
Const
    FBAUDRATE      = 1;
    FPARITY         = 2;
    FCHARSIZE       = 4;
    FHANDSHAKE      = 5;
    FCAUTOLF        = $11;
    Baud2400        = 3;

```



```

ParDisabled      = 0;
CharSz8          = 0;
XonXoff          = 6;
Type
String64         = String [64];
pString64        = ^String64;
StrTbl           = Array [1..3] Of String64;
pStrTbl          = Array [1..3] Of pString64;
Var
  Args:           StrTbl;
  Adrs:           pStrTbl;
  Result:         Integer;
  Arg:            ^pStrTbl;
  Dump:           pString80;
Begin {LoadDriver}
  IO_Unit := OSdcmlDv;
  Arg := @Adrs;
  Adrs [1] := @Args [1];
  Adrs [2] := @Args [2];
  Adrs [3] := @Args [3];
  Args [1] := ^/ccsys/drv.dtacom^;
  Args [2] := ^DTACOM1^;
  Args [3] := ^31^;
  Dump := pOSdevNam (IO_Unit);
  Dump := ^;
  If 0 = Call (^Assign^, Input, Output, Arg^, 3)
  Then
    Begin
      UnitClear (IO_Unit);
      Result := Baud2400;
      UnitStatus (IO_Unit, Result, FBAUDRATE);
      Result := ParDisabled;
      UnitStatus (IO_Unit, Result, FPARITY);
      Result := XonXoff;
      UnitStatus (IO_Unit, Result, FHANDSHAKE);
      Result := CharSz8;
      UnitStatus (IO_Unit, Result, FCHARSIZE);
      UnitStatus (IO_Unit, Result, FCAUTOLF);
      LoadDriver := True
    End
  Else
    Begin
      IO_Unit := -1;
      LoadDriver := False
    End
  {EndIf}
End; {LoadDriver}
Function Init_HP7221 ((Colour: ScreenColour;
                     Style: LineType): Boolean);
Begin {Init_HP7221}
  If LoadDriver
  Then
    Begin
      SegLock;
      Mode := Mode None;
      Colour_HP7221 (Colour);
      Mode := Mode None;
      Style_HP7221 (Style);
      Left := X_Min_Port;
      Right := X_Max_Port;
      Bottom := Y_Min_Port;
      Top := Y_Max_Port;
      Buffer_Pointer := 0;
      Mode := Mode None;
      Old X := 0;
      Old Y := 0;
      Init_HP7221 := True;
      {Turn on plotter.}
      {These are escape characters...}
      ...
    End
  End

```

```

        Stuff (^B.(BMO;0;0;13;0:B.I800;^);
        {These are escape characters...}
        Stuff (^O;17:B.NO;19:B.@i110;0: ^);
        {Set machine limits.}
        Set_HP7221 (Left, Right, Bottom, Top);
    End
Else
    Init_HP7221 := False
{EndIf}
End; {Init_HP7221}
Procedure Inq_HP7221 ((Var XMin, XMax, YMin, YMax,
                    XDotsPerInch, YDotsPerInch: Integer));
Begin {Inq_HP7221}
    XMin := Left;
    XMax := Right;
    YMin := Bottom;
    YMax := Top;
    XDotsPerInch := X Dots Millimeter;
    YDotsPerInch := Y Dots Millimeter;
End; {Inq_HP7221}
{$S}
{EndImplementation}
End.

```

Appendix A-6 - IoWindow Unit

```

Unit IoWindow;
Interface
  Uses
    { $U /CCUTIL/CCLIB }
    CCdefn,
    CCwndIO;
  Function WindResult: Integer;
  Procedure Select Root Window;
  Procedure Select Command Window;
  Procedure InitWindow;
{End Interface}
Implementation
  Var
    IOstat: Integer;
  Function WindResult { : Integer };
  Begin {WindResult}
    WindResult := IOstat
  End; {WindResult}
  Procedure Select Root Window;
  Begin {Select Root Window}
    IOstat := WinSystem (WsysCurr)
  End; {Select Root Window}
  Procedure Select Command Window;
  Begin {Select Command Window}
    IOstat := WinSystem (WsysCmd)
  End; {Select Command Window}
  Procedure InitWindow;
  Begin {InitWindow}
    CCwndIOinit
  End; {InitWindow}
{End Implementation}
End.

```

Appendix A-7 - Create Program

```

Program Create;
Const
  Char_Width      = 4;
  Char_Height     = 6;
  Char_Spacing    = 5;
  Max_Letters     = 127;
  Max_Coords      = 500;
Type
  String80        = String [80];
  String255       = String [255];
  Type_Byte       = -128..127;
  Type_Coord      = Type_Byte;
Var
  Letters:        String255;
  Ptrs:           Array [0..Max_Letters] Of Integer;
  Coords:         Array [1..Max_Coords, 1..2] Of Type_Coord;
Procedure Init_Soft;
Var
  Next_Loc:       Integer;
Procedure Init;
Begin
  Letters := '';
  Next_Loc := 1;
End;
Procedure New_Letter (Ch: Char);
Begin
  Letters := Concat (Letters, Ch);
  Letters[Length (Letters)] := Ch;
  Ptrs[Length (Letters)] := Next_Loc;
End;
Procedure Draw (X, Y: Type_Coord);
Begin
  Coords[Next_Loc, 1] := X;
  Coords[Next_Loc, 2] := Y;
  Next_Loc := Next_Loc + 1;
End;
Procedure Slew (X, Y: Type_Coord);
Begin
  Draw (X + Char_Spacing, Y);
End;
Procedure Term;
Const
  Name_Out      = 'Graphic_Letters';
Var
  Last:         Integer;
  File_Out:     Interactive;
  I:            Integer;
  J:            Integer;
Begin
  Last := Length (Letters);
  Ptrs [Last + 1] := Next_Loc;
  Rewrite (File_Out, Name_Out);
  Writeln (File_Out, Name_Out, ' opened. ');
  Writeln (File_Out, Char_Width: 4,
           Char_Height: 4, Char_Spacing: 4);
  Writeln (File_Out, 'Character - width: ', Char_Width: 1, ' height: ',
           Char_Height: 1, ' spacing: ', Char_Spacing: 1);
  For I := 1 To Last Do
    Begin
      Writeln (File_Out, 'Character ', Letters [I], ' has ',
              Ptrs [I + 1] - Ptrs [I]: 1, ' coordinates. ');
      Writeln (File_Out, Letters [I],
              Ptrs [I + 1] - Ptrs [I]: 4);
      For J := Ptrs [I] To Ptrs [I + 1] - 1 Do
        If Coords [J, 1] >= Char_Spacing
          Then

```

```

        Writeln (File_Out, 'S', Coords [J, 1] -
                Char_Spacing: 4, Coords [J, 2]: 4)
    Else
        Writeln (File_Out, 'D', Coords [J, 1]: 4,
                Coords [J, 2]: 4)
    {EndIf}
{EndFor}
End;
{EndFor}
Close (File_Out, Lock);
Writeln ('File: ', Name_Out, ' closed. ');
End;
Begin
Init;

New Letter ('A');
Slew (0, 0); Draw (0, 5); Draw (1, 6); Draw (3, 6);
Draw (4, 5); Draw (4, 0); Slew (0, 3); Draw (4, 3);

New Letter ('B');
Slew (0, 0); Draw (0, 6); Draw (3, 6); Draw (4, 5);
Draw (4, 4); Draw (3, 3); Draw (4, 2); Draw (4, 1);
Draw (3, 0); Draw (0, 0); Slew (0, 3); Draw (3, 3);

New Letter ('C');
Slew (4, 5); Draw (3, 6); Draw (1, 6); Draw (0, 5);
Draw (0, 1); Draw (1, 0); Draw (3, 0); Draw (4, 1);

New Letter ('D');
Slew (0, 0); Draw (3, 0); Draw (4, 1); Draw (4, 5);
Draw (3, 6); Draw (0, 6); Draw (0, 0);

New Letter ('E');
Slew (4, 0); Draw (0, 0); Draw (0, 6); Draw (4, 6);
Slew (0, 3); Draw (3, 3);

New Letter ('F');
Slew (0, 0); Draw (0, 6); Draw (4, 6); Slew (0, 3);
Draw (3, 3);

New Letter ('G');
Slew (3, 3); Draw (4, 3); Draw (4, 1); Draw (3, 0);
Draw (1, 0); Draw (0, 1); Draw (0, 5); Draw (1, 6);
Draw (3, 6); Draw (4, 5);

New Letter ('H');
Slew (0, 0); Draw (0, 6); Slew (0, 3); Draw (4, 3);
Slew (4, 6); Draw (4, 0);

New Letter ('I');
Slew (0, 0); Draw (4, 0); Slew (2, 0); Draw (2, 6);
Slew (0, 6); Draw (4, 6);

New Letter ('J');
Slew (0, 1); Draw (1, 0); Draw (3, 0); Draw (4, 1);
Draw (4, 6); Draw (2, 6);

New Letter ('K');
Slew (0, 0); Draw (0, 6); Slew (4, 6); Draw (0, 2);
Slew (2, 4); Draw (4, 0);

New Letter ('L');
Slew (0, 6); Draw (0, 0); Draw (4, 0);

New Letter ('M');
Slew (0, 0); Draw (0, 6); Draw (2, 0); Draw (4, 6);
Draw (4, 0);

New Letter ('N');
Slew (0, 0); Draw (0, 6); Draw (4, 0); Draw (4, 6);

```

```
New Letter (^O^);
SleW {0, 1}; Draw {1, 0}; Draw {3, 0}; Draw {4, 1};
Draw {4, 5}; Draw {3, 6}; Draw {1, 6}; Draw {0, 5};
Draw {0, 1};
```

```
New Letter (^P^);
SleW {0, 0}; Draw {0, 6}; Draw {3, 6}; Draw {4, 5};
Draw {4, 4}; Draw {3, 3}; Draw {0, 3};
```

```
New Letter (^Q^);
SleW {0, 1}; Draw {1, 0}; Draw {3, 0}; Draw {4, 1};
Draw {4, 5}; Draw {3, 6}; Draw {1, 6}; Draw {0, 5};
Draw {0, 1}; Slew {3, 1}; Draw {4, 0};
```

```
New Letter (^R^);
SleW {0, 0}; Draw {0, 6}; Draw {3, 6}; Draw {4, 5};
Draw {4, 4}; Draw {3, 3}; Draw {0, 3}; Slew {3, 3};
Draw {4, 0};
```

```
New Letter (^S^);
SleW {0, 1}; Draw {1, 0}; Draw {3, 0}; Draw {4, 1};
Draw {4, 2}; Draw {3, 3}; Draw {1, 3}; Draw {0, 4};
Draw {0, 5}; Draw {1, 6}; Draw {3, 6}; Draw {4, 5};
```

```
New Letter (^T^);
SleW {2, 0}; Draw {2, 6}; Slew {0, 6}; Draw {4, 6};
```

```
New Letter (^U^);
SleW {0, 6}; Draw {0, 1}; Draw {1, 0}; Draw {3, 0};
Draw {4, 1}; Slew {4, 6}; Draw {4, 0};
```

```
New Letter (^V^);
SleW {0, 6}; Draw {2, 0}; Draw {4, 6};
```

```
New Letter (^W^);
SleW {0, 6}; Draw {1, 0}; Draw {2, 3}; Draw {3, 0};
Draw {4, 6};
```

```
New Letter (^X^);
SleW {0, 6}; Draw {4, 0}; Slew {0, 0}; Draw {4, 6};
```

```
New Letter (^Y^);
SleW {0, 6}; Draw {2, 3}; Draw {4, 6}; Slew {2, 3};
Draw {2, 0};
```

```
New Letter (^Z^);
SleW {0, 6}; Draw {4, 6}; Draw {0, 0}; Draw {4, 0};
Slew {1, 3}; Draw {3, 3};
```

```
New Letter (^O^);
SleW {0, 1}; Draw {1, 0}; Draw {3, 0}; Draw {4, 1};
Draw {4, 5}; Draw {3, 6}; Draw {1, 6}; Draw {0, 5};
Draw {0, 1}; Slew {0, 0}; Draw {4, 6};
```

```
New Letter (^1^);
SleW {0, 0}; Draw {4, 0}; Slew {2, 0}; Draw {2, 6};
Draw {0, 4};
```

```
New Letter (^2^);
SleW {0, 5}; Draw {1, 6}; Draw {3, 6}; Draw {4, 5};
Draw {4, 4}; Draw {3, 3}; Draw {1, 3}; Draw {0, 2};
Draw {0, 0}; Draw {4, 0};
```

```
New Letter (^3^);
SleW {0, 1}; Draw {1, 0}; Draw {3, 0}; Draw {4, 1};
Draw {4, 2}; Draw {3, 3}; Draw {4, 4}; Draw {4, 5};
Draw {3, 6}; Draw {1, 6}; Draw {0, 5}; Slew {1, 3};
Draw {3, 3};
```

```

New Letter (^4^);
Slew {1, 6}; Draw {0, 3}; Draw {4, 3}; Slew {3, 6};
Draw {2, 0};

New Letter (^5^);
Slew {0, 1}; Draw {1, 0}; Draw {3, 0}; Draw {4, 1};
Draw {4, 2}; Draw {3, 3}; Draw {0, 3}; Draw {1, 6};
Draw {4, 6};

New Letter (^6^);
Slew {0, 2}; Draw {1, 3}; Draw {3, 3}; Draw {4, 2};
Draw {4, 1}; Draw {3, 0}; Draw {1, 0}; Draw {0, 1};
Draw {0, 5}; Draw {1, 6}; Draw {3, 6}; Draw {4, 5};

New Letter (^7^);
Slew {2, 0}; Draw {4, 6}; Draw {1, 6}; Draw {0, 5};
Slew {2, 3}; Draw {4, 3};

New Letter (^8^);
Slew {1, 3}; Draw {0, 4}; Draw {0, 5}; Draw {1, 6};
Draw {3, 6}; Draw {4, 5}; Draw {4, 4}; Draw {3, 3};
Draw {4, 2}; Draw {4, 1}; Draw {3, 0}; Draw {1, 0};
Draw {0, 1}; Draw {0, 2}; Draw {1, 3}; Draw {3, 3};

New Letter (^9^);
Slew {0, 1}; Draw {1, 0}; Draw {3, 0}; Draw {4, 1};
Draw {4, 5}; Draw {3, 6}; Draw {1, 6}; Draw {0, 5};
Draw {0, 4}; Draw {1, 3}; Draw {3, 3}; Draw {4, 4};

New Letter (^+^);
Slew {0, 3}; Draw {4, 3}; Slew {2, 1}; Draw {2, 5};

New Letter (^-^);
Slew {0, 3}; Draw {4, 3};

New Letter (^.^);
Slew {0, 0}; Draw {1, 0}; Draw {1, 1}; Draw {0, 1};
Draw {0, 0};

New Letter (^,^);
Slew {0, -1}; Draw {1, 0}; Draw {1, 1}; Draw {0, 1};
Draw {0, 0};

New Letter (^;^);
Slew {0, -1}; Draw {1, 0}; Draw {1, 1}; Draw {0, 1};
Draw {0, 0}; Slew {0, 2}; Draw {0, 3}; Draw {1, 3};
Draw {1, 2}; Draw {0, 2};

New Letter (^:~);
Slew {0, 0}; Draw {1, 0}; Draw {1, 1}; Draw {0, 1};
Draw {0, 0}; Slew {0, 2}; Draw {0, 3}; Draw {1, 3};
Draw {1, 2}; Draw {0, 2};

New Letter (^(^);
Slew {2, 0}; Draw {1, 1}; Draw {1, 5}; Draw {2, 6};

New Letter (^)~);
Slew {2, 0}; Draw {3, 1}; Draw {3, 5}; Draw {2, 6};

New Letter (^([~);
Slew {3, 0}; Draw {1, 0}; Draw {1, 6}; Draw {3, 6};

New Letter (^]~);
Slew {1, 0}; Draw {3, 0}; Draw {3, 6}; Draw {1, 6};

New Letter (^=^);
Slew {0, 2}; Draw {4, 2}; Slew {4, 4}; Draw {0, 4};

Term

```

```
End;  
Begin  
  WriteLn ('Graphic character file creation commences...');  
  Init Soft;  
  WriteLn ('Graphic character file creation terminates...')  
End.
```


Appendix B - CifDraw IC Layout Program Software Listings

Appendix B-1 - LayoutDefs Unit

```

Unit LayoutDefs;
Interface
Const
    LambdaXextent = 320;    {Compartment X extent in lambda.}
    LambdaYextent = 320;    {Compartment Y extent in lambda.}
    Lambda        = 18;     {Number of lambdas per inch.}
    LambdaToGamma = 2;
    X Compartments = 30;    {Compartments in X direction.}
    Y Compartments = 30;    {Compartments in Y direction.}
    Name In       = ^/SYSTEM; {For blind i/o.}
    LambdaGrid    = 10;     {Spacing between grid points.}
    Codes         = 'ABCDEFGHJKLMNPOQRSTUVWXYZabcdefghijklmn';
    Asc ESC       = 'B';
    SysTerm       = 2;
    String Length = 8;
    Command_Length = 16;
    Init          = $ff;
    SetKey        = $fe;
    TurnOff       = $fd;
    TurnOn        = $fc;
    MaxRect       = 10;

    {Define tokens to be returned by the Decode function...}

    US_F01 = 0;    {Unshift Functions}
    US_F02 = 1;
    US_F03 = 2;
    US_F04 = 3;
    US_F05 = 4;
    US_F06 = 5;
    US_F07 = 6;
    US_F08 = 7;
    US_F09 = 8;
    US_F10 = 9;

    SH_F01 = 10;   {Shift Functions}
    SH_F02 = 11;
    SH_F03 = 12;
    SH_F04 = 13;
    SH_F05 = 14;
    SH_F06 = 15;
    SH_F07 = 16;
    SH_F08 = 17;
    SH_F09 = 18;
    SH_F10 = 19;

    CUS_F01 = 20;  {Command Unshift Functions}
    CUS_F02 = 21;
    CUS_F03 = 22;
    CUS_F04 = 23;
    CUS_F05 = 24;
    CUS_F06 = 25;
    CUS_F07 = 26;
    CUS_F08 = 27;
    CUS_F09 = 28;
    CUS_F10 = 29;

    CSH_F01 = 30;  {Command Shift Functions}
    CSH_F02 = 31;
    CSH_F03 = 32;
    CSH_F04 = 33;
    CSH_F05 = 34;
    CSH_F06 = 35;
    CSH_F07 = 36;
    CSH_F08 = 37;
    CSH_F09 = 38;

```

```

CSH_F10          = 39;
CSR_UP           = 40; {Cursor Keys}
CSR_DOWN         = 41;
CSR_RIGHT        = 42;
CSR_LEFT         = 43;

Type
XCompartments    = 0 .. X Compartments;
YCompartments    = 0 .. Y Compartments;
TypeLayer        = (PW, F, PG, NG, CP, P, NP, PP, C, M, G);
TypeCoord        = Integer;
TypeMeasure      = Integer;
TypeDirection    = Integer;
TypeObject       = Record
    Xcenter:      TypeCoord;
    Ycenter:      TypeCoord;
    Width:        TypeMeasure;
    Lngth:        TypeMeasure;
End;

pCompRect        = ^CompRect;
CompRect         = Record
    NextBlock:    pCompRect;
    Contents:     Array [1..MaxRect]
                  Of TypeObject
End;

Rect2Type        = Record
    NoOfEntries:  Integer;
    FirstRect:    pCompRect
End;

RectType         = Record
    Residents:    Set Of TypeLayer;
    Headers:      Array [TypeLayer]
                  Of Rect2Type
End;

ExtType          = Record
    X_Min:        TypeMeasure;
    X_Max:        TypeMeasure;
    Y_Min:        TypeMeasure;
    Y_Max:        TypeMeasure
End;

pPrimExtent      = ^PrimExtent;
PrimExtent       = Record
    Extents:      ExtType;
    RectDescriptor: RectType
End;

CompInfo         = pPrimExtent;

TypeName         = Packed Array [1..String_Length] Of Char;
TypeCommand      = String [Command_Length];

Var
{The layout space data structure...}

Compartments:    Array [Xcompartments, Ycompartments]
                  Of CompInfo;

{
  +-----+ <-- WindowTop
  | Window |
  +-----+ <-- WindowBottom
  |         |
  |         | <-- WindowRight
  |         | <-- WindowLeft
}

WindowLeft:      TypeCoord;
WindowRight:     TypeCoord;
WindowBottom:    TypeCoord;
WindowTop:       TypeCoord;

```

{World coordinate boundaries.}

```

WorldLeft:      TypeCoord;
WorldRight:     TypeCoord;
WorldBottom:    TypeCoord;
WorldTop:       TypeCoord;

{Number of lambdas for current device.}

X_Gamma:        Integer;
Y_Gamma:        Integer;

X_Left:         Xcompartments;
X_Right:        Xcompartments;
Y_Bottom:       Ycompartments;
Y_Top:          Ycompartments;

CursorX:        TypeCoord;      {Center of cursor.}
CursorY:        TypeCoord;
X_Old:          TypeCoord;
Y_Old:          TypeCoord;

StepCursor:     TypeCoord;      {Cursor movement factor.}

JumpScreen:     Real;
SlowSlew:       Integer;
FastSlew:       Integer;

Gamma:          Integer;

Magnification:  Real;
Mag_Num:        Integer;
Mag_Den:        Integer;

LayersVisible:  Set Of TypeLayer;
LayerCurrent:   TypeLayer;      {Layer being defined.}

{Bottom of heap before dynamic allocation.}
HeapBottom:     Integer;

Free_Comp:      pCompRect;

GridSpacing:    Integer;        {In gamma units.}
Grid_On:        Boolean;

{Blind i/o channel to keyboard.}
File_In:        Interactive;

In_Char:        Char;

Saved:          Boolean;         {Work space saved.}
Dirty:          Boolean;         {Work space is dirty.}

Parameter:      Record
    Key_Number:  Integer;
    Key_String:  TypeName;
    Key_Command: TypeCommand;
End;

{EndInterface}
Implementation
{EndImplementation}
End. {DataDefs Unit}

```

Appendix B-2 - CifDraw Program

Program Layout:

```

Uses
  {SU /ccutil/GraphLib.Obj}
  IoWindow,
  {SU /ccutil/GraphLib.Obj}
  GraphDefs,
  {SU /ccutil/GraphLib.Obj}
  Graph,
  {SU LayoutDefs.Obj}
  LayoutDefs,
  {SU /ccutil/cclib.obj}
  CcDirIO;
{SSSeg 01}
Procedure Set Menu; Forward;
Procedure Lock_2; Forward;
Procedure Lock_3; Forward;
Procedure Lock_4; Forward;
Procedure Lock_5; Forward;
Procedure Lock_1;
Begin
  Lock_2
End;
Procedure Install (KeyNumber: Integer; KeyString: TypeName);
Begin
  Parameter.Key Number := KeyNumber;
  Parameter.Key String := KeyString;
  Parameter.Key Command := 'B#';
  Parameter.Key Command [3] := Chr (KeyNumber + Ord ('A'));
  UnitStatus (System, Parameter, SetKey)
End;
Function AskNumber (Msg: String255; Low, High: Integer): Integer;
Var
  Response: Integer;
Begin {AskNumber}
  Repeat
    Write (Msg, ' (', Low: 1, '...', High: 1, '): ');
    Erase Line;
    Readln (Response);
    If (Low > Response) Or (High < Response)
    Then
      Writeln ('*** Invalid response (',
        Response: 1, ').' ***', Chr (7))
    {EndIf}
  Until (Low <= Response) And (Response <= High);
  AskNumber := Response
End; {AskNumber}
Procedure StatusContext;
Const
  Layers = ' PW F PG NG CP P NP PP C M G ';
Var
  I: TypeLayer;
Begin
  Select Command Window;
  Move Cursor (46, 0);
  For I := PW To G Do
    If I = LayerCurrent
    Then
      Begin
        Inverse Video;
        Write (Copy (Layers, Ord (I) * 4 + 1, 4))
      End
    Else
      Begin
        Normal Video;
        Write (' ')
      End
    End;

```

```

        {EndIf}
      {EndFor}
      Move Cursor (46, 1);
      For T := PW To G Do
        Begin
          If I In LayersVisible
            Then
              Inverse_Video
            Else
              Normal_Video;
          {EndIf}
          Write (Copy (Layers, Ord (I) * 4 + 1, 4))
        End;
      {EndFor}
      Inverse_Video;
      Select_Root_Window
    End;
  Procedure StatusCursor;
  Begin {StatusCursor}
    Select Command Window;
    Move Cursor (12, 0);
    Write (CursorX Div LambdaToGamma: 6, ' ');
    Move Cursor (25, 0);
    Write (CursorY Div LambdaToGamma: 6, ' ');
    Move Cursor (12, 1);
    Write (Abs (CursorX - X_Old) Div LambdaToGamma: 6, ' ');
    Move Cursor (25, 1);
    Write (Abs (CursorY - Y_Old) Div LambdaToGamma: 6, ' ');
    Select Root Window
  End; {StatusCursor}
  Procedure Status;
  Begin {Status}
    Select Command Window;
    Normal_Video;
    Erase Screen;
    Move Cursor (0, 0);
    Write ('Cursor X -           Y -           Current -');
    Move Cursor (0, 1);
    Write ('Delta X -           Y -           Visible -');
    Inverse_Video;
    Select_Root_Window;
    StatusCursor;
    StatusContext
  End; {Status}
  Function Decode: Integer;
  Const
    Asc_ESC      = 'B';
    Asc_Leadin   = '#';
    Error        = -1;
  Var
    In_Char:      Char;
  Begin {Decode}
    Read (File In, In_Char);
    If In_Char = Asc_ESC
      Then
        Begin
          Read (File In, In_Char);
          If In_Char = Asc_Leadin
            Then
              Begin
                Read (File In, In_Char);
                If (In_Char >= 'A') And (In_Char <= 'h')
                  Then
                    Decode := Ord (In_Char) - Ord ('A')
                  Else
                    Decode := Error
                  {EndIf}
                End
              End
            Else
              Begin

```

```

        If (In_Char >= 'A') And (In_Char <= 'D')
        Then
            Decode := Ord (In_Char) - Ord ('A') + 40
        Else
            Decode := Error
        {EndIf}
    End
{EndIf}
End
Else
    Decode := Error
{EndIf}
End; {Decode}
Function Alc_PrimExtent (Var Result: pPrimExtent): Boolean;
Var
    Temp: TypeLayer;
Begin
    New (Result);
    If Result <> Nil
    Then
        With Result^ Do
            Begin
                Extents.X_Min := 32000;
                Extents.X_Max := 0;
                Extents.Y_Min := 32000;
                Extents.Y_Max := 0;
                RectDescriptor.Residents := [];
                For Temp := PW To G Do
                    Begin
                        RectDescriptor.Headers [Temp].NoOfEntries := 0;
                        RectDescriptor.Headers [Temp].FirstRect := Nil
                    End
                {EndFor}
            End;
        {EndWith}
    {EndIf}
    Alc_PrimExtent := Result <> Nil {I love obtuse code!}
End; {Alc_PrimExtent}
Function Alc_CompRect (Var Result: pCompRect): Boolean;
Begin
    If Free_Comp <> Nil
    Then
        Begin
            Result := Free_Comp;
            Free_Comp := Free_Comp^.NextBlock
        End
    Else
        New (Result);
    {EndIf}
    If Result <> Nil
    Then
        Begin
            Result^.NextBlock := Nil;
            Alc_CompRect := True;
        End
    Else
        Alc_CompRect := False
    {EndIf}
End; {Alc_CompRect}
Function Alc_Room (Layer: TypeLayer; X_Ptr: Xcompartments;
                  Y_Ptr: Ycompartments): Boolean;
Var
    Ptr_PrimExtent: pPrimExtent;
    Ptr_CompRect: pCompRect;
    Temp_CompRect: pCompRect;
Begin
    Alc_Room := False; {Guilty till proven innocent!}
    If Compartments [X_Ptr, Y_Ptr] = Nil
    Then {Need to allocate a primary extent.}
        If Alc_PrimExtent (Ptr_PrimExtent)

```

```

Then
  If Alc_CompRect (Ptr_CompRect)
  Then
    Begin
      With Ptr_PrimExtent^.RectDescriptor Do
        Begin
          Residents := [Layer];
          Headers [Layer].FirstRect := Ptr_CompRect
        End;
      {EndWith}
      Compartments [X_Ptr, Y_Ptr] := Ptr_PrimExtent;
      Alc_Room := True
    End
  Else
    Writeln ('<Alc_Room> - no comprects.')
  {EndIf}
Else
  Writeln ('<Alc_Room> - no primextents.')
{EndIf}
Else
  With Compartments [X_Ptr, Y_Ptr]^ .RectDescriptor Do
    If Layer In Residents
    Then
      If Headers [Layer].NoOfEntries Mod MaxRect = 0
      Then
        If Alc_CompRect (Ptr_CompRect)
        Then
          Begin
            Temp_CompRect := Headers [Layer].FirstRect;
            While Temp_CompRect^.NextBlock <> Nil Do
              Temp_CompRect := Temp_CompRect^.NextBlock;
            {EndWhile}
            Temp_CompRect^.NextBlock := Ptr_CompRect;
            Alc_Room := True
          End
        Else
          Writeln ('<Alc_Room> - no comprects.')
        {EndIf}
      Else
        Alc_Room := True
      {EndIf}
    Else
      If Alc_CompRect (Ptr_CompRect)
      Then
        Begin
          Headers [Layer].FirstRect := Ptr_CompRect;
          Residents := Residents + [Layer];
          Alc_Room := True
        End
      Else
        Writeln ('<Alc_Room> - no comprects.')
      {EndIf}
    {EndIf}
  {EndWith}
{EndIf}
End; {Alc_Compartment}
Function Delete_Object (Layer: TypeLayer; X_Center,
  Y_Center: TypeCoord; X_Size,
  Y_Size: TypeMeasure): Boolean;
Var
  X_Addr: Xcompartments;
  Y_Addr: Ycompartments;
  Delete_Point: Integer;
  Ptr_Temp: pCompRect;
  Ptr_Last: pCompRect;
  Count: Integer;
  Source: Integer;
  Destination: Integer;
  Finished: Boolean;
  Found: Boolean;

```



```

Ptr_CompRect:      pCompRect;
L:                 TypeLayer;
I:                 Integer;
X_L:               Integer;
X_R:               Integer;
Y_B:               Integer;
Y_T:               Integer;
Begin
  X_Addr := X_Center Div LambdaXextent;
  Y_Addr := Y_Center Div LambdaYextent;
  Found := False;
  If Compartments [X_Addr, Y_Addr] <> Nil
  Then
    With Compartments [X_Addr, Y_Addr]^
      RectDescriptor.Headers [Layer] Do
        If NoOfEntries <> 0
        Then
          Begin
            Ptr_Temp := FirstRect;
            Delete_Point := 1;
            Finished := False;
            Repeat
              With Ptr_Temp^.Contents [(Delete_Point - 1) Mod
                MaxRect + 1] Do
                If (Xcenter = X_Center) And
                  (Ycenter = Y_Center) And
                  (Width = X_Size) And
                  (Length = Y_Size)
                Then
                  Found := True
                Else
                  Begin
                    Delete_Point := Delete_Point + 1;
                    If Delete_Point > NoOfEntries
                    Then
                      Finished := True
                    Else
                      If (Delete_Point - 1) Mod
                        MaxRect = 0
                      Then
                        Begin
                          Ptr_Temp := Ptr_Temp^.
                            NextBlock;
                          If Ptr_Temp = Nil
                          Then
                            Finished := True
                          {EndIf}
                        End
                      {EndIf}
                    {EndIf}
                  End
                {EndIf}
              {EndWith}
            Until Found Or Finished;
            If Found
            Then
              Begin
                Ptr_Last := Ptr_Temp;
                While Ptr_Last^.NextBlock <> Nil Do
                  Ptr_Last := Ptr_Last^.NextBlock;
                {EndWhile}
                If Delete_Point <> NoOfEntries
                Then
                  Begin
                    Source := (NoOfEntries - 1) Mod
                      MaxRect + 1;
                    Destination := (Delete_Point - 1) Mod
                      MaxRect + 1;
                    Ptr_Temp^.Contents [Destination].
                      Xcenter := Ptr_Last^.

```

```

        Contents [Source].Xcenter;
    Ptr Temp^.Contents [Destination].
        Ycenter := Ptr Last^.
        Contents [Source].Ycenter;
    Ptr Temp^.Contents [Destination].
        Width := Ptr Last^.
        Contents [Source].Width;
    Ptr Temp^.Contents [Destination].
        Length := Ptr Last^.
        Contents [Source].Length;
End;
{EndIf}
If NoOfEntries Mod MaxRect = 1
Then
    If NoOfEntries = 1
    Then
        Begin
            Ptr Temp^.NextBlock := Free_Comp;
            Free_Comp := Ptr Temp;
            FirstRect := Nil;
            NoOfEntries := 0;
            With Compartments [X Addr, Y Addr]^
            RectDescriptor Do
                Residents := Residents - [Layer]
            {EndWith}
        End
    Else
        Begin
            If Ptr_Temp = Ptr_Last
            Then
                Ptr Temp := FirstRect;
            {EndIf}
            While Ptr Temp^.NextBlock <>
                Ptr_Last Do
                Ptr Temp := Ptr_Temp^.NextBlock;
            {EndWhile}
            Ptr_Last^.NextBlock := Free_Comp;
            Free_Comp := Ptr Temp;
            Ptr Temp^.NextBlock := Nil;
            NoOfEntries := NoOfEntries - 1
        End
    {EndIf}
    Else
        NoOfEntries := NoOfEntries - 1
    {EndIf}
End
{EndIf}
End;
{EndIf}
{EndWith}
{EndIf}
If Found
Then
    With Compartments [X Addr, Y Addr]^ Do
        If RectDescriptor.Residents = []
        Then
            With Extents Do
                Begin
                    X_Min := 32000;
                    X_Max := 0;
                    Y_Min := 32000;
                    Y_Max := 0
                End
            {EndWith}
        Else
            Begin
                With RectDescriptor Do
                    Begin
                        X_L := 32000;
                        X_R := 0;

```

```

Y_B := 32000;
Y_T := 0;
For L := PW To G Do
  If L In Residents
  Then
    Begin
      Ptr_CompRect := Headers [L].
      FirstRect;
      For I := 0 To Headers [L].
        NoOfEntries - 1 Do
          Begin
            With Ptr_CompRect^. Contents
              [(I Mod MaxRect) + 1] Do
                Begin
                  X_L := Min (X_L, Xcenter
                    - Width Div 2);
                  X_R := Max (X_R, Xcenter
                    + Width Div 2);
                  Y_B := Min (Y_B, Ycenter
                    - Lngt Div 2);
                  Y_T := Max (Y_T, Ycenter
                    + Lngt Div 2)
                End;
              {EndWith}
            If (I + 1) Mod MaxRect = 0
            Then
              Ptr_CompRect :=
                Ptr_CompRect^.NextBlock
            {EndIf}
          End
        {EndFor}
      End
    {EndIf}
  {EndFor}
End;
{EndWith}
With Extents Do
  Begin
    X_Min := X_L;
    X_Max := X_R;
    Y_Min := Y_B;
    Y_Max := Y_T
  End
{EndWith}
End;
{EndIf}
{EndWith}
{EndIf}
Delete Object := Found
End; {Delete Object}
Function Insert Object (Layer: TypeLayer; X_Center, Y_Center:
  TypeCoord; X_Size, Y_Size: TypeMeasure): Boolean;
Var
  X_Addr: Xcompartments;
  Y_Addr: Ycompartments;
  Success: Boolean;
  Insert Point: Integer;
  Ptr_Temp: pCompRect;
Begin
  Dirty := True;
  X_Addr := X_Center Div LambdaXextent;
  Y_Addr := Y_Center Div LambdaYextent;
  Success := Alc_Room (Layer, X_Addr, Y_Addr);
  If Success
  Then
    With Compartments [X_Addr, Y_Addr]^ Do
      Begin
        With RectDescriptor.Headers [Layer] Do
          Begin
            Ptr_Temp := FirstRect;

```

```

While Ptr Temp^.NextBlock <> Nil Do
  Ptr Temp := Ptr Temp^.NextBlock;
{EndWhile}
With Ptr Temp^.Contents [(NoOfEntries Mod
  MaxRect) + 1] Do
  Begin
    Xcenter := X Center;
    Ycenter := Y Center;
    Width := X Size;
    Length := Y Size;
  End;
{EndWith}
NoOfEntries := NoOfEntries + 1
End;
{EndWith}
With Extents Do
  Begin
    X Min := Min (X Min, X Center - X Size Div 2);
    X Max := Max (X Max, X Center + X Size Div 2);
    Y Min := Min (Y Min, Y Center - Y Size Div 2);
    Y Max := Max (Y Max, Y Center + Y Size Div 2)
  End;
{EndWith}
End;
{EndWith}
Insert Object := Success
End; {Insert Object}
Procedure NewWindow (X_Corner, Y_Corner: Integer);
Begin {NewWindow}
  WindowLeft := X Corner;
  WindowBottom := Y Corner;
  WindowRight := WindowLeft + Round (Magnification * X Gamma);
  WindowTop := WindowBottom + Round (Magnification * Y Gamma);
  SetWindow (WindowLeft, WindowRight, WindowBottom, WindowTop)
End; {NewWindow}
Procedure SlewHairs (X_Coord, Y_Coord: Integer);
Const
  Offset = 4; {Gamma units}
Var
  OldColour: ScreenColour;
Begin {SlewHairs}
  InqColour (OldColour);
  SetColour (Flip);
  SetLine (Solid);
  SlewAbs (X Coord - Offset, Y Coord);
  DrawAbs (X Coord + Offset, Y Coord);
  SlewAbs (X Coord, Y Coord - Offset);
  DrawAbs (X Coord, Y Coord + Offset);
  SetLine (Dotted);
  SetColour (OldColour)
End; {SlewHairs}
Procedure MarkHairs (X_Coord, Y_Coord: Integer);
Const
  Offset = 4; {Gamma units}
Var
  OldColour: ScreenColour;
Begin {MarkHairs}
  InqColour (OldColour);
  SetColour (Flip);
  SetLine (Solid);
  SlewAbs (X Coord - Offset, Y Coord);
  DrawAbs (X Coord, Y Coord - Offset);
  DrawAbs (X Coord + Offset, Y Coord);
  DrawAbs (X Coord, Y Coord + Offset);
  DrawAbs (X Coord - Offset, Y Coord);
  SetLine (Dotted);
  SetColour (OldColour)
End; {MarkHairs}
Procedure DrawGrid;

```

```

Var
  X_Start:      Integer;
  Y_Start:      Integer;
  X_Temp:       Integer;
  X_Bound:      Integer;
  Y_Bound:      Integer;
  W_Left:       Integer;
  W_Bottom:     Integer;
Begin {DrawGrid}
  If Grid_On
  Then
    Begin
      W_Left := Max (WindowLeft, WorldLeft);
      X_Start := W_Left Mod GridSpacing;
      If X_Start = 0
      Then
        X_Start := W_Left
      Else
        X_Start := W_Left + GridSpacing - X_Start;
      {EndIf}
      W_Bottom := Max (WindowBottom, WorldBottom);
      Y_Start := W_Bottom Mod GridSpacing;
      If Y_Start = 0
      Then
        Y_Start := W_Bottom
      Else
        Y_Start := W_Bottom + GridSpacing - Y_Start;
      {EndIf}
      X_Bound := Min (WorldRight, WindowRight);
      Y_Bound := Min (WorldTop, WindowTop);
      Repeat
        X_Temp := X_Start;
        Repeat
          PointAbs (X_Temp, Y_Start);
          X_Temp := X_Temp + GridSpacing
        Until X_Temp > X_Bound;
        Y_Start := Y_Start + GridSpacing
      Until Y_Start > Y_Bound
    End
  {EndIf}
End; {DrawGrid}
Procedure SetLayerColour (Layer: TypeLayer);
Begin
  Case Layer Of
    PW:
      Begin
        SetColour (Brown);
        SetLine (Solid)
      End;
    F:
      Begin
        SetColour (Blue);
        SetLine (Solid)
      End;
    PG:
      Begin
        SetColour (Yellow);
        SetLine (Solid)
      End;
    NG:
      Begin
        SetColour (Green);
        SetLine (Solid)
      End;
    CP:
      Begin
        SetColour (Brown);
        SetLine (Solid)
      End;
    P:

```

```

        Begin
            SetColour (Orange);
            SetLine (Solid)
        End;
    NP:
        Begin
            SetColour (Red);
            SetLine (Solid)
        End;
    PP:
        Begin
            SetColour (Blue);
            SetLine (Dotted)
        End;
    C:
        Begin
            SetColour (Green);
            SetLine (Solid)
        End;
    M:
        Begin
            SetColour (Black);
            SetLine (Solid)
        End;
    G:
        Begin
            SetColour (Blue);
            SetLine (Solid)
        End
    End
    {EndCase}
End;
{$SSeg_02}
Procedure Lock_2;
Begin
    Lock_3
End;
Procedure Draw_Rectangle (X, Y: TypeCoord;
                          Lngth, Width: TypeMeasure);
Begin
    SlewAbs (X - Lngth Div 2, Y - Width Div 2);
    DrawRel (Lngth, 0);
    DrawRel (Lngth, Width);
    DrawRel (0, Width);
    DrawRel (0, 0)
End; {Draw_Rectangle}

Procedure Plot;
Const
    X_Viewport_Bias = 0.05;
    Y_Viewport_Bias = 0.05;
Var
    OldColour:      ScreenColour;
    X_Page_Size:    Integer;
    Y_Page_Size:    Integer;
    X_Min_Layout:   Integer;
    X_Max_Layout:   Integer;
    Y_Min_Layout:   Integer;
    Y_Max_Layout:   Integer;
    Delta_X:        Integer;
    Delta_Y:        Integer;
    Finished:       Boolean;
    Device_Left:    Integer;
    Device_Right:   Integer;
    Device_Bottom:  Integer;
    Device_Top:     Integer;
    X_Dots:         Integer;
    Y_Dots:         Integer;
Procedure Set_Page (X_Size, Y_Size: Real);
Const

```

```

    InchToMillimeter := 25.4;
    Slop := 10; {millimeters}
    Begin {Set Page}
        X_Page_Size := Round (X_Size * InchToMillimeter) - Slop; {mm}
        Y_Page_Size := Round (Y_Size * InchToMillimeter) - Slop; {mm}
        Device_Left := 0;
        Device_Right := X_Page_Size * X_Dots;
        Device_Bottom := 0;
        Device_Top := Y_Page_Size * Y_Dots;
        SetDevice (Device_Left, Device_Right,
                  Device_Bottom, Device_Top);
        Erase Screen;
        WriteLn ('Page size - X: ', X_Size: 1: 1,
                ' Y: ', Y_Size: 1: 1)
    End; {Set Page}
    Procedure Master_Menu;
    Begin
        UnitStatus (System, Parameter, Init);
        Install (SH_F01, '8.5x11" ');
        Install (US_F01, 'Page ');
        Install (SH_F02, '11x8.5" ');
        Install (US_F02, 'Page ');
        Install (SH_F03, '14x8.5" ');
        Install (US_F03, 'Page ');
        Install (SH_F04, '16.5x11" ');
        Install (US_F04, 'Page ');
        Install (US_F06, 'Start ');
        Install (SH_F10, 'Exit ');
        UnitStatus (System, Parameter, TurnOn)
    End;
    Function Plot_Init: Boolean;
    Var
        Initialized: Boolean;
    Begin
        InqColour (OldColour);
        TermGraphics; {On concept}
        Select Command_Window;
        Curse On;
        Normal Video;
        Erase Screen;
        If InItGraphics (HP7221)
        Then
            Begin
                InqDevice (Device_Left, Device_Right, Device_Bottom,
                          Device_Top, X_Dots, Y_Dots);
                Set Page (8.5, 11.0);
                Plot_Init := True;
                Master_Menu
            End
        Else
            Plot_Init := False
        {EndIf}
    End;
    Procedure Plot_Stuff;
    Var
        X_Min_Marker: Integer;
        X_Max_Marker: Integer;
        Y_Min_Marker: Integer;
        Y_Max_Marker: Integer;
        Correct_Colour: Boolean;
        Ptr_CompRect: pCompRect;
        I: Integer;
        Layer: TypeLayer;
        X_Ptr: Integer;
        Y_Ptr: Integer;
        Got_Hit: Boolean;
        Finished: Boolean;
    Procedure Alignment (X_Left, X_Right, Y_Bottom, Y_Top: Integer);
    Procedure Cross (X, Y: Integer);
    Begin

```

```

        SlewAbs (X + 0, Y + 1);
        DrawAbs (X + 2, Y + 1);
        SlewAbs (X + 1, Y + 0);
        DrawAbs (X + 1, Y + 2);
    End;
Begin {Alignment}
    SetColour (Black);
    SetLine (Solid);
    SetWindow (0, 6, 0, 6);
    SetViewport (0.0, 2.0 * X Viewport Bias, 0.0,
                2.0 * Y Viewport Bias);
        Cross (0, 2);
        Cross (2, 0);
    SetViewport (1.0 - 2.0 * X Viewport Bias, 1.0,
                0.0, 2.0 * Y Viewport Bias);
        Cross (2, 0);
        Cross (4, 2);
    SetViewport (0.0, 2.0 * X Viewport Bias,
                1.0 - 2.0 * Y Viewport Bias, 1.0);
        Cross (0, 2);
        Cross (2, 4);
    SetViewport (1.0 - 2.0 * X Viewport Bias, 1.0,
                1.0 - 2.0 * Y Viewport Bias, 1.0);
        Cross (2, 4);
        Cross (4, 2);
    End; {Alignment}
Begin {Plot Stuff}
    UnitStatus (System, Parameter, Init);
    Install (US F06, 'Continue');
    Install (SH F10, 'Exit');
    Erase Screen;
    X_Min_Layout := AskNumber ('Lower X Coordinate', 0,
                               WorldRight Div LambdaToGamma);
    Y_Min_Layout := AskNumber ('Lower Y Coordinate', 0,
                               WorldTop Div LambdaToGamma);
    Erase Screen;
    X_Max_Layout := AskNumber ('Upper X Coordinate',
                               X_Min_Layout, WorldRight Div
                               LambdaToGamma);
    Y_Max_Layout := AskNumber ('Upper Y Coordinate',
                               Y_Min_Layout, WorldTop Div
                               LambdaToGamma);
    Erase Screen;
    X_Min_Layout := X_Min_Layout * LambdaToGamma;
    X_Max_Layout := X_Max_Layout * LambdaToGamma;
    Y_Min_Layout := Y_Min_Layout * LambdaToGamma;
    Y_Max_Layout := Y_Max_Layout * LambdaToGamma;
{***} Delta_X := Trunc (((1.0 - 2.0 * X Viewport Bias) *
                        X Page Size * Gamma * Magnification) /
                        25.4);
{***} Delta_Y := Trunc (((1.0 - 2.0 * Y Viewport Bias) *
                        Y Page Size * Gamma * Magnification) /
                        25.4);
    {Make sure page is integral cif size.}
    Delta_X := Delta_X + Delta_X Mod LambdaToGamma;
    Delta_Y := Delta_Y + Delta_Y Mod LambdaToGamma;
    Finished := False;
    X_Min_Marker := X_Min_Layout;
    While (X_Min_Marker < X_Max_Layout) And Not Finished Do
        Begin
            X_Max_Marker := X_Min_Marker + Min (Delta_X,
                                                  X_Max_Layout - X_Min_Marker);
            Y_Min_Marker := Y_Min_Layout;
            While (Y_Min_Marker < Y_Max_Layout) And
                Not Finished Do
                Begin
                    Y_Max_Marker := Y_Min_Marker + Min (Delta_Y,
                                                          Y_Max_Layout - Y_Min_Marker);
                    Erase Screen;
                    Write ('Coordinates for this page - ');

```



```

Write  (('(', X Min Marker Div LambdaToGamma: 1,
        ')', Y Min Marker Div LambdaToGamma: 1,
        ') to ');
Writeln (('(', X Max Marker Div LambdaToGamma: 1,
        ')', Y Max Marker Div LambdaToGamma: 1,
        ')');
Write
('Hit <Continue> to plot it, <Exit> to quit. ');
UnitStatus (System, Parameter, TurnOn);
UnitClear (System);
Got Hit := False;
Repeat
  Case Decode Of
    US F06:
      Got Hit := True;
    SH F10:
      Begin
        Got Hit := True;
        Finished := True
      End;
    Otherwise:
      Ring_Bell
  End
{EndCase}
Until Got Hit;
Erase Screen;
UnitStatus (System, Parameter, TurnOff);
If Not Finished
Then
  Begin
    Alignment (X Min Marker, X Max Marker,
               Y Min Marker, Y Max Marker);
    SetWindow (X Min Marker, X Max Marker,
               Y Min Marker, Y Max Marker);
    SetViewport (X Viewport Bias,
                 X Viewport Bias + ((1.0 - 2.0
                                     * X Viewport Bias) *
                                     (X Max Marker - X Min Marker)
                                     ) / Delta X,
                 Y Viewport Bias,
                 Y Viewport Bias + ((1.0 - 2.0
                                     * Y Viewport Bias) *
                                     (Y Max Marker - Y Min Marker)
                                     ) / Delta Y);
    For Layer := PW To G DO
      If Layer In LayersVisible
      Then
        Begin
          Correct Colour := False;
          For X Ptr := 0 To X Compartments Do
            For Y Ptr := 0 To Y Compartments Do
              If Compartments [X_Ptr, Y_Ptr] <> Nil
              Then
                With Compartments [X_Ptr,
                                   Y_Ptr] Do
                  IF (Extents.X Min <=
                      X Max Marker)
                  And (Extents.X Max >=
                      X Min Marker)
                  And (Extents.Y Min <=
                      Y Max Marker)
                  And (Extents.Y Max >=
                      Y Min Marker)
                  And (Layer In
                      RectDiscriptor.
                      Residents)
                  Then
                    With RectDiscriptor Do
                      If Headers [Layer].NoOfEntries > 0
                      Then

```

```

Begin
  If Not Correct_Colour
  Then
    Begin
      SetLayerColour (Layer);
      Correct_Colour := True
    End;
  {EndIf}
  Ptr CompRect :=
  Headers [Layer].FirstRect;
  For I := 0 To Headers [Layer].
  NoOfEntries - 1 Do
    Begin
      With Ptr CompRect.
        Contents[(I Mod MaxRect
        ) + 1] Do
          Draw Rectangle (
            Xcenter, Ycenter,
            Width, Lngth);
        {EndWith}
        If (I + 1) Mod MaxRect=0
        Then
          Ptr CompRect :=
          Ptr CompRect.
          NextBlock
        {EndIf}
      End
    {EndFor}
  End
  {EndIf}
  {EndWith}
  {EndIf}
  {EndFor}
  {EndFor}
  End;
  {EndFor}
  NewPage;
  Y_Min_Marker := Y_Max_Marker
End
{EndIf}
  End;
  {EndWhile}
  X_Min_Marker := X_Max_Marker
End;
{EndWhile}
Master Menu
End; {Plot Stuff}
Procedure Plot_Term;
Var
  Junk: Boolean;
Begin {Plot Term}
  TermGraphics; {On HP plotter}
  Erase Screen;
  Curse Off;
  Select Root Window;
  Junk := InitGraphics (Concept);
  InqDevice (Device Left, Device_Right, Device_Bottom,
  Device Top, X Dots, Y Dots);
  Device_Right := Device Left + X Gamma * X Dots Div Gamma;
  Device_Top := Device Bottom + Y Gamma * Y Dots Div Gamma;
  SetDevice (Device Left, Device_Right, Device_Bottom,
  Device_Top);
  SetWindow (WindowLeft, WindowRight, WindowBottom,
  WindowTop);
  SetColour (OldColour);
  SetLine (Dotted);
  Set Menu;
  Status

```



```

FirstRect;
For I := 0 To Headers [Layer].
  NoOfEntries - 1 Do
  Begin
    With Ptr CompRect^.Contents
      [(I Mod MaxRect) + 1] Do
        Draw Rectangle (Xcenter,
          Ycenter, Width, Lngth);
      {EndWith}
      If (I + 1) Mod MaxRect = 0
      Then
        Ptr CompRect :=
          Ptr CompRect^.NextBlock
      {EndIf}
    End
  {EndFor}
  End
  {EndWith}
  {EndIf}
  {EndWith}
  {EndIf}
  {EndFor}
  {EndFor}
  End;
  {EndIf}
  {EndFor}
  SetLine (Dotted);
  End;
  Procedure ScreenAction;
  Begin {ScreenAction}
    SetWindow (WindowLeft, WindowRight, WindowBottom, WindowTop);
    RedrawScreen;
    MarkHairs (X_Old, Y_Old);
  End; {ScreenAction}
  Procedure New Screen;
  Begin {New Screen}
    ScreenAction;
    SlewHairs (CursorX, CursorY)
  End;
  Procedure Position (X_Coord, Y_Coord: TypeCoord);
  Var
    Delta X:          TypeCoord;
    Delta Y:          TypeCoord;
  Begin {Position}
    SlewHairs (CursorX, CursorY);
    Delta X := Round (Magnification * X_Gamma) Div 2;
    Delta Y := Round (Magnification * Y_Gamma) Div 2;
    If X_Coord - Delta_X < WorldLeft
    Then
      Begin
        WindowLeft := WorldLeft;
        WindowRight := WindowLeft + 2 * Delta_X
      End
    Else
      If X_Coord + Delta_X > WorldRight
      Then
        Begin
          WindowRight := WorldRight;
          WindowLeft := WindowRight - 2 * Delta_X
        End
      Else
        Begin
          WindowLeft := X_Coord - Delta_X;
          WindowRight := X_Coord + Delta_X
        End
      End;
    {EndIf}
  {EndIf}
  If Y_Coord - Delta_Y < WorldBottom
  Then
    Begin

```

```

If 0 <> Pos ('DS', Buffer)
Then
  Begin
    Finished := Eof (In_File);
    Count := 0;
    Erase Screen;
    While Not Finished Do
      Begin
        Case In_File^ Of
          'L':
            Begin
              Get (In_File);
              Get (In_File);
              Case In_File^ Of
                'P':
                  Begin
                    Get (In_File);
                    Case In_File^ Of
                      'W': Layer := PW;
                      'G': Layer := PG;
                      'P': Layer := PP;
                      ':': Layer := P;
                    End;
                  End;
                'N':
                  Begin
                    Get (In_File);
                    Case In_File^ Of
                      'C': Layer := NC;
                      'P': Layer := NP;
                    End;
                  End;
                'C':
                  Begin
                    Get (In_File);
                    Case In_File^ Of
                      'P': Layer := CP;
                      ':': Layer := C;
                    End;
                  End;
                'P':
                  Begin
                    Get (In_File);
                    Layer := F;
                  End;
                'M':
                  Begin
                    Get (In_File);
                    Layer := M;
                  End;
                'G':
                  Begin
                    Get (In_File);
                    Layer := G;
                  End;
                Otherwise:
                  ; {just in case}
                End;
              {EndCase}
            End;
          'B':
            Begin
              Get (In_File);
              Count := Count + 1;
              Move Cursor (0, 0);
              WriteLn (':Rectangles read: ',
                Count);
              Read (In_File, X Size, Y Size,
                X Center, Y Center);
              If Not Insert_Object (Layer,

```

```

Then
  Ring_Bell
Else
  Begin
    SlewHairs (CursorX, CursorY);
    CursorX := CursorX + StepCursor;
    If CursorX > WindowRight
    Then
      ScreenRight;
    {EndIf}
    SlewHairs (CursorX, CursorY)
  End;
{EndIf}
StatusCursor
End; {CursorRight}
Procedure CursorDown;
Begin {CursorDown}
  If CursorY - StepCursor < WorldBottom
  Then
    Ring_Bell
  Else
    Begin
      SlewHairs (CursorX, CursorY);
      CursorY := CursorY - StepCursor;
      If CursorY < WindowBottom
      Then
        ScreenDown;
      {EndIf}
      SlewHairs (CursorX, CursorY)
    End;
  {EndIf}
  StatusCursor
End; {CursorDown}
Procedure CursorUp;
Begin {CursorUp}
  If CursorY + StepCursor > WorldTop
  Then
    Ring_Bell
  Else
    Begin
      SlewHairs (CursorX, CursorY);
      CursorY := CursorY + StepCursor;
      If CursorY > WindowTop
      Then
        ScreenUp;
      {EndIf}
      SlewHairs (CursorX, CursorY)
    End;
  {EndIf}
  StatusCursor
End; {CursorUp}
Procedure Slow Slew;
{Fix this to lock onto an appropriate coord.}
Begin {Slow Slew}
  StepCursor := SlowSlew * LambdaToGamma
End; {Slow Slew}
Procedure Fast Slew;
{Fix this to lock onto an appropriate coord.}
Begin {Fast Slew}
  SlewHairs (CursorX, CursorY); {Turn off the dregs.}
  CursorX := Min (WorldRight, ((CursorX + GridSpacing Div 2) Div
    GridSpacing) * GridSpacing);
  CursorY := Min (WorldTop, ((CursorY + GridSpacing Div 2) Div
    GridSpacing) * GridSpacing);
  SlewHairs (CursorX, CursorY);
  StepCursor := FastSlew * LambdaToGamma;
  StatusCursor
End; {Fast Slew}
{$SSeg_03}
Procedure Lock_3;

```

```

Begin
  Lock_4
End;
Procedure On_Grid;
Var
  OldColour:          ScreenColour;
Begin {On_Grid}
  InqColour (OldColour);
  SetColour (White);
  Grid On := True;
  DrawGrid;
  SetColour (OldColour)
End; {On_Grid}
Procedure Off_Grid;
Var
  OldColour:          ScreenColour;
Begin {Off_Grid}
  InqColour (OldColour);
  SetColour (Flip);
  Grid On := True;
  DrawGrid; {Erase the existing grid}
  Grid On := False;
  SetColour (OldColour)
End; {Off_Grid}
Procedure Set_Location;
Var
  X_Coord:          TypeCoord;
  Y_Coord:          TypeCoord;
Begin
  Select Command_Window;
  Normal_Video;
  Erase_Screen;
  Curse-On;
  X_Coord := AskNumber ('X Coordinate', 0, WorldRight Div 2);
  Y_Coord := AskNumber ('Y Coordinate', 0, WorldTop Div 2);
  Curse Off;
  Erase_Screen;
  Select Root_Window;
  Position (X_Coord * 2, Y_Coord * 2);
  Status
End; {Set_Location}
Procedure Write_File;
Type
  String30 = String [30];
Var
  Ptr CompRect:      pCompRect;
  X_Ptr:             Xcompartments;
  Y_Ptr:             Ycompartments;
  I:                 Integer;
  Layer:             TypeLayer;
  Out_Name:          String30;
  Out_File:          Interactive;
  FinIshed:          Boolean;
  Error:             Integer;
  Count:             Integer;
Procedure Abort (Fid: String30; Error: Integer);
Begin
  Erase_Screen;
  WriteIn ('*** Cannot save file ', Fid,
    ' due to I/O error ', Error: 1, ' ***', Chr (7));
  Write ('<<Carriage return to continue>');
  Readln;
  Curse Off;
  Erase_Screen;
  Select Root_Window;
  Status;
  Exit (Write_File)
End;
Begin {File Write}
  {Change To write out by layer, not by compartment.}

```

```

Select Command_Window;
Normal_Video;
Erase_Screen;
Curse_On;
Saved := False;
Write ('File to save CIF symbol in: ');
ReadLn (Out_Name);
Curse_Off;
If Length (Out_Name) <> 0
Then
  Begin
    {SI-}
    Rewrite (Out_File, Out_Name);
    {SI+}
    Error := IoResult;
    If Error <> 0
    Then
      Abort (Out_Name, Error);
    {EndIf}
    {SI-}
    Writeln (Out_File, 'DS Carleton University CIF File!');
    {SI+}
    Error := IoResult;
    If Error <> 0
    Then
      Abort (Out_Name, Error);
    {EndIf}
    Count := 0;
    Erase_Screen;
    For X_Ptr := 0 To X_Compartments Do
      For Y_Ptr := 0 To Y_Compartments Do
        If Compartments [X_Ptr, Y_Ptr] <> Nil
        Then
          With Compartments [X_Ptr, Y_Ptr]^
          RectDescriptor Do
            For Layer := PW To G Do
              If Layer In Residents
              Then
                Begin
                  {SI-}
                  Writeln (Out_File, 'LC',
                    Copy ('PWF PGNGCPP NPPPC M G ', 2 *
                      Ord (Layer) + 1, 2), ' ');
                  {SI+}
                  Error := IoResult;
                  If Error <> 0
                  Then
                    Abort (Out_Name, Error);
                  {EndIf}
                  Ptr CompRect := Headers [Layer].
                    FirstRect;
                  For I := 0 To Headers [Layer].
                    NoOfEntries - 1 Do
                    Begin
                      With Ptr CompRect^
                      Contents [(I Mod MaxRect) + 1]
                      Do
                        {SI-}
                        Writeln (Out_File, 'B',
                          Width: 5, Length: 5,
                          Xcenter: 5, Ycenter: 5,
                          ' ');
                        {SI+}
                      {EndWith}
                      Count := Count + 1;
                      Move_Cursor (0, 0);
                      Writeln ('Rectangles written: ',
                        Count);
                      Error := IoResult;
                      If Error <> 0

```



```

Then
    Abort (Out_Name, Error);
{EndIf}
If (I + 1) Mod MaxRect = 0
Then
    Ptr CompRect := Ptr_CompRect^;
    NextBlock
{EndIf}
End
{EndFor}
End;
{EndIf}
{EndWith}
{EndIf}
{EndFor}
{EndFor}
{SI-}
WriteLn (Out_File, 'DF;');
{SI+}
Error := IoResult;
If Error <> 0
Then
    Abort (Out_Name, Error);
{EndIf}
Saved := True;
Dirty := False;
Close (Out_File, Lock)
End
Else
    Saved := True;
{EndIf}
Erase Screen;
Select_Root_Window;
Status
End; {File Write}
Procedure Read_File;
Var
    X_Center:      TypeCoord;
    Y_Center:      TypeCoord;
    X_Size:         TypeMeasure;
    Y_Size:         TypeMeasure;
    X_Dir:          TypeDirection;
    Y_Dir:          TypeDirection;
    Layer:          TypeLayer;
    In_Name:        String [30];
    In_File:        Interactive;
    Buffer:          String [80];
    Finished:       Boolean;
    Count:          Integer;
Begin
    Select_Command_Window;
    Normal_Video;
    Erase Screen;
    Cursor_On;
    Write('File to read CIF symbol from: ');
    ReadLn (In_Name);
    Cursor_Off;
    If Length (In_Name) <> 0
    Then
        Begin
            {SI-}
            Reset (In_File, In_Name);
            {SI+}
            If IoResult <> 0
            Then
                WriteLn ('*** Unable to open ', In_Name, '. ***',
                    Chr (7))
            Else
                Begin
                    ReadLn (In_File, Buffer);

```

```

If 0 <> Pos ('DS', Buffer)
Then
  Begin
    Finished := Eof (In_File);
    Count := 0;
    Erase Screen;
    While Not Finished Do
      Begin
        Case In_File Of
          'L':
            Begin
              Get (In_File);
              Get (In_File);
              Case In_File Of
                'P':
                  Begin
                    Get (In_File);
                    Case In_File Of
                      'W': Layer := PW;
                      'G': Layer := PG;
                      'P': Layer := PP;
                      ':': Layer := P;
                    End;
                  End;
                'N':
                  Begin
                    Get (In_File);
                    Case In_File Of
                      'G': Layer := NG;
                      'P': Layer := NP;
                    End;
                  End;
                'C':
                  Begin
                    Get (In_File);
                    Case In_File Of
                      'P': Layer := CP;
                      ':': Layer := c;
                    End;
                  End;
                'F':
                  Begin
                    Get (In_File);
                    Layer := F;
                  End;
                'M':
                  Begin
                    Get (In_File);
                    Layer := M;
                  End;
                'G':
                  Begin
                    Get (In_File);
                    Layer := G;
                  End;
                Otherwise:
                  ; {just in case}
              End;
            EndCase)
          End;
        'B':
          Begin
            Get (In_File);
            Count := Count + 1;
            Move Cursor (0, 0);
            WriteLn ('Rectangles read: ',
              Count);
            Read (In_File, X Size, Y Size,
              X Center, Y Center);
            If Not Insert_Object (Layer,

```

```

CursorX + X_Center, CursorY
+ Y_Center, X_Size, Y_Size)
Then
    Finished := True
{EndIf}
End;
'D':
    Finished := True;
Otherwise:
    ; {just in case}
End;
{EndCase}
ReadLn (In_File);
Finished := Finished Or Eof (In_File)
End
{EndWhile}
Else
    Writeln ('*** ', In_Name,
            ' is not a CIF file. ***', Chr (7))
{EndIf}
End;
{EndIf}
Close (In_File, Lock)
End;
{EndIf}
Erase Screen;
Select Root Window;
New Screen;
Status
End; {Read File}
Procedure Prologue;
Var
    X_Ptr:           XCompartments;
    Y_Ptr:           YCompartments;
    X_Dots:          Integer;
    Y_Dots:          Integer;
    ScreenLeft:      Integer;
    ScreenRight:     Integer;
    ScreenBottom:    Integer;
    ScreenTop:       Integer;
    Junk:            Boolean;
Begin {Prologue}
    Junk := InitGraphics (Concept);
    InitWindow;
    Select Command Window;
    Erase Screen;
    Curse Off;
    Select Root Window;
    For X_Ptr := 0 To X Compartments Do
        For Y_Ptr := 0 To Y Compartments Do
            Compartments [X_Ptr, Y_Ptr] := Nil;
        {EndFor}
    {EndFor}
    Free Comp := Nil;
    Gamma := LambdaToGamma * Lambda;
    InqDevice (ScreenLeft, ScreenRight, ScreenBottom, ScreenTop,
               X_Dots, Y_Dots);
    X_Gamma := (ScreenRight - ScreenLeft) * Gamma Div X_Dots;
    Y_Gamma := (ScreenTop - ScreenBottom) * Gamma Div Y_Dots;
    ScreenRight := ScreenLeft + X_Gamma * X_Dots Div Gamma;
    ScreenTop := ScreenBottom + Y_Gamma * Y_Dots Div Gamma;
    SetDevice (ScreenLeft, ScreenRight, ScreenBottom, ScreenTop);
    Mag Num := 1;
    Mag Den := 1;
    Magnification := 1.0;
    NewWindow (0, 0);
    GridSpacing := LambdaGrid * LambdaToGamma;
    Grid On := True;
    SlowSlew := 1;

```

```

FastSlew := LambdaGrid;
StepCursor := LambdaToGamma * FastSlew;
CursorX := 0;
CursorY := 0;
X Old := 0;
Y Old := 0;
JumpScreen := 0.5;
X Left := 0;
X Right := 0;
Y Bottom := 0;
Y Top := 0;
WorldLeft := 0;
WorldBottom := 0;
WorldRight := LambdaXextent * X Compartments;
WorldTop := LambdaYextent * Y Compartments;
LayersVisible := [PW .. G];
LayerCurrent := PW;
RedrawScreen;
Reset (File In, Name In);
SetColour (White); {As opposed to Xor lines}
SlewHairs (CursorX, CursorY);
MarkHairs (X Old, Y Old);
SetLine (Dotted);
Saved := False; {Work space hasn't been saved.}
Dirty := False; {Nothing has been defined.}
Status;
End; {Prologue}
Procedure Efficiency;
Var
  X Ptr:                      Xcompartments;
  Y Ptr:                      Ycompartments;
  Layer:                      TypeLayer;
  Entries:                    LongInt;
  Blocks:                    LongInt;
Begin {Efficiency}
  Erase Screen;
  Write ('+-----+');
  Writeln ('+-----+');
  For Layer := PW To G Do
    Begin
      Entries := 0;
      Blocks := 0;
      For X Ptr := 0 To X Compartments Do
        For Y Ptr := 0 To Y Compartments Do
          If Compartments [X_Ptr, Y_Ptr] <> Nil
            Then
              With Compartments [X_Ptr, Y_Ptr]^
                RectDescriptor Do
                  If Layer In Residents
                    Then
                      Begin
                        Entries := Entries + Headers [Layer].
                        NoOfEntries;
                        Blocks := Blocks + (Headers [Layer].
                        NoOfEntries + MaxRect - 1) Div MaxRect
                      End;
                    {EndIf}
                  {EndIf}
                {EndIf}
              {EndFor}
            {EndFor}
          {EndFor}
        Writeln ('| Layer - ', Ord (Layer), : 2, ' | Entries - ',
        Entries: 10, ' | Blocks - ', Blocks: 10,
        ' | Utilization - ', Entries * 100.0 /
        (Blocks * MaxRect): 6: 2, '% |');
        Write ('+-----+');
        Writeln ('+-----+');
      End;
    {EndFor}
  Writeln;

```

```

Write ('<Return> to continue.~');
Readln;
New Screen
End; {Efficiency}
Procedure Dump_Space;
Var
    X_Ptr:      Xcompartments;
    Y_Ptr:      Ycompartments;
    I:          Integer;
    Ptr_CompRect: pCompRect;
    Layer:      TypeLayer;
    Ghost:      Record
        Case Integer Of
            1:(Rect_Ptr:      pCompRect);
            2:(Prim_Ptr:      pPrimExtent);
            3:(Address:      LongInt)
        End;
Begin
    Erase Screen;
    WriteIn (~-----+~);
    Writeln (~| Memory Available: ~, MemAvail: 10,
              |~);
    Writeln (~-----+~);
    Writeln (~|X_Ptr|Y_Ptr|Xcenter|Ycenter|Width|Lngh|~);
    Writeln (~-----+~);
    For Layer := PW To G Do
        For X_Ptr := 0 To X Compartments Do
            For Y_Ptr := 0 To Y Compartments Do
                If Compartments [X_Ptr, Y_Ptr] <> Nil
                    Then
                        With Compartments [X_Ptr, Y_Ptr]^ . RectDiscriptor
                            Do
                                If (Layer In Residents) And (Layer In
                                    LayersVisible)
                                    Then
                                        Begin
                                            Ptr_CompRect := Headers [Layer].FirstRect;
                                            WriteIn (~|Number Of Entries: ~,
                                                    |Headers [Layer].NoOfEntries: 8,
                                                    |Layer: ~,Ord (layer): 2,
                                                    |~);
                                            Write (~-----+~);
                                            Writeln (~-----+~);
                                            Ghost.Rect_Ptr := Ptr_CompRect;
                                            While Ptr_CompRect <> Nil Do
                                                Begin
                                                    Ghost.Rect_Ptr := Ptr_CompRect;
                                                    WriteIn (~|Current Address: ~,
                                                            |Ghost.Address: 10,
                                                            |~);
                                                    Write (~-----+~);
                                                    Write (~-----+~);
                                                    Writeln;
                                                    For I := 1 To MaxRect Do
                                                        With Ptr_CompRect^.Contents [I]
                                                            Do
                                                                Writeln (~|X_Ptr: 5,~|,
                                                                    |Y_Ptr: 5,~|,
                                                                    |Xcenter: 8,~|,
                                                                    |Ycenter: 8,~|,
                                                                    |Width: 8,~|,
                                                                    |Lngh: 8,~-};
                                                                {EndWith}
                                                            {EndFor}
                                                        Write (~-----+~);
                                                        Writeln(~-----+~);
                                                        Ptr_CompRect := Ptr_CompRect^.
                                                            NextBlock
                                                    End
                                                {EndWhile}
                    End
                End
            End
        End
    End

```

```

                                End;
                                {EndIf}
                                {EndWith}
                                {EndIf}
                                {EndFor}
                                {EndFor}
                                {EndFor}
                                {EndFor}
                                WriteLn;
                                Write ('<Return> to continue. ');
                                ReadLn;
                                New_Screen
End;
Procedure Set_Menu;
Begin
  UnitStatus (System, Parameter, Init);
  Install (US_F01, 'Mark');
  Install (SH_F01, 'Delete');
  Install (US_F02, '% Effect');
  Install (US_F03, 'Redraw');
  Install (SH_F05, 'Set');
  Install (US_F05, 'Current');
  Install (US_F06, 'Grid On');
  Install (SH_F06, 'Grid Off');
  Install (US_F07, 'Goto');
  Install (US_F08, 'SlewSlow');
  Install (US_F09, 'SlewFast');
  Install (US_F10, 'Slew');
  Install (CSH_F01, 'Set');
  Install (CUS_F01, 'Visible');
  Install (CSH_F02, 'ZERO');
  Install (CUS_F02, 'Plot');
  Install (CSH_F03, 'Set');
  Install (CUS_F03, 'Mag');
  Install (CSH_F04, 'Set');
  Install (CUS_F04, 'SlowSlew');
  Install (CSH_F05, 'Set');
  Install (CUS_F05, 'FastSlew');
  Install (CSH_F06, 'Set');
  Install (CUS_F06, 'ScrnSlew');
  Install (CSH_F07, 'Set Grid');
  Install (CUS_F07, 'ListFile');
  Install (CSH_F08, 'SaveFile');
  Install (CUS_F08, 'Get File');
  Install (CSH_F09, 'Or Line');
  Install (CUS_F09, 'Xor Line');
  Install (CUS_F10, 'Exit');
  Install (CSH_F10, '>-Dump-<');
  UnitStatus (System, Parameter, TurnOn);
  UnitClear (System);
End;
Procedure Set_Magnification;
Var
  Finished: Boolean;
Begin
  UnitStatus (System, Parameter, Init);
  Install (US_F01, '8 X');
  Install (US_F02, '4 X');
  Install (US_F03, '2 X');
  Install (US_F04, '1 X');
  Install (US_F05, '1/2 X');
  Install (US_F06, '1/3 X');
  Install (US_F07, '1/4 X');
  Install (US_F08, '1/5 X');
  Install (US_F09, '1/6 X');
  Install (US_F10, '1/10 X');
  UnitStatus (System, Parameter, TurnOn);
  UnitClear (System);
  Mag_Num := 1;
  Mag_Den := 1;
  Repeat

```

```

Finished := True;
Case Decode Of
  US_F01:
    Mag_Den := 8;
  US_F02:
    Mag_Den := 4;
  US_F03:
    Mag_Den := 2;
  US_F04:
    Mag_Den := 1;
  US_F05:
    Mag_Num := 2;
  US_F06:
    Mag_Num := 3;
  US_F07:
    Mag_Num := 4;
  US_F08:
    Mag_Num := 5;
  US_F09:
    Mag_Num := 6;
  US_F10:
    Mag_Num := 10;
Otherwise:
  Begin
    Ring_Bell;
    Finished := False;
  End
End
{EndCase}
Until Finished;
Magnification := Mag_Num / Mag_Den;
WindowLeft := CursorX - Round (Magnification * X_Gamma *
(CursorX - WindowLeft) / (WindowRight - WindowLeft));
WindowRight := WindowLeft + Round (Magnification * X_Gamma *
(WindowRight - WindowLeft) / (WindowRight - WindowLeft));
WindowBottom := CursorY - Round (Magnification * Y_Gamma *
(CursorY - WindowBottom) / (WindowTop - WindowBottom));
WindowTop := WindowBottom + Round (Magnification * Y_Gamma *
(WindowTop - WindowBottom) / (WindowTop - WindowBottom));
If Mag_Num >= 4
Then
  Grid_On := False;
{EndIf}
New_Screen;
Set_Menu;
End;
{$SSeg 04}
Procedure Lock_4;
Begin
  Lock_5;
End;
Procedure Set_Slow_Slew;
Var
  Finished: Boolean;
Begin
  UnitStatus (System, Parameter, Init);
  Install (US_F01, -1 lambda);
  Install (US_F02, -2 lambda);
  Install (US_F03, -3 lambda);
  Install (US_F04, -4 lambda);
  Install (US_F05, -5 lambda);
  Install (US_F06, -10 lambda);
  Install (US_F07, -15 lambda);
  Install (US_F08, -20 lambda);
  Install (US_F09, -30 lambda);
  Install (US_F10, -40 lambda);
  UnitStatus (System, Parameter, TurnOn);
  UnitClear (System);
  Repeat
    Finished := True;
    Case Decode Of
      US_F01:

```

```

        SlowSlew := 1;
US F02:
    ~SlowSlew := 2;
US F03:
    ~SlowSlew := 3;
US F04:
    ~SlowSlew := 4;
US F05:
    ~SlowSlew := 5;
US F06:
    ~SlowSlew := 10;
US F07:
    ~SlowSlew := 15;
US F08:
    ~SlowSlew := 20;
US F09:
    ~SlowSlew := 30;
US F10:
    ~SlowSlew := 40;
Otherwise:
    Begin
        Ring Bell;
        Finished := False
    End
End
{EndCase}
Until Finished;
Slow Slew;
Set Menu
End;
Procedure Set_Fast_Slew;
Var
    Finished:      Boolean;
Begin
    UnitStatus (System, Parameter, Init);
    Install (US F01, ~ 1 lambd);
    Install (US F02, ~ 2 lambd);
    Install (US F03, ~ 3 lambd);
    Install (US F04, ~ 4 lambd);
    Install (US F05, ~ 5 lambd);
    Install (US F06, ~10 lambd);
    Install (US F07, ~15 lambd);
    Install (US F08, ~20 lambd);
    Install (US F09, ~30 lambd);
    Install (US F10, ~40 lambd);
    UnitStatus (System, Parameter, TurnOn);
    UnitClear (System);
    Repeat
        Finished := True;
        Case Decode Of
            US F01:
                ~FastSlew := 1;
            US F02:
                ~FastSlew := 2;
            US F03:
                ~FastSlew := 3;
            US F04:
                ~FastSlew := 4;
            US F05:
                ~FastSlew := 5;
            US F06:
                ~FastSlew := 10;
            US F07:
                ~FastSlew := 15;
            US F08:
                ~FastSlew := 20;
            US F09:
                ~FastSlew := 30;
            US F10:
                ~FastSlew := 40;

```



```

        Otherwise:
            Begin
                Ring_Bell;
                Finished := False
            End
        End
    {EndCase}
Until Finished;
Fast_Slew;
Set_Menu
End;
Procedure Set_Jump_Screen;
Var
    Finished: Boolean;
Begin
    UnitStatus (System, Parameter, Init);
    Install (US_F01, '4/4 scrn');
    Install (US_F02, '3/4 scrn');
    Install (US_F03, '2/4 scrn');
    Install (US_F04, '1/4 scrn');
    UnitStatus (System, Parameter, TurnOn);
    UnitClear (System);
    Repeat
        Finished := True;
        Case Decode Of
            US_F01:
                JumpScreen := 1;
            US_F02:
                JumpScreen := 3 / 4;
            US_F03:
                JumpScreen := 1 / 2;
            US_F04:
                JumpScreen := 1 / 4;
            Otherwise:
                Begin
                    Ring_Bell;
                    Finished := False
                End
            End
        {EndCase}
    Until Finished;
    Set_Menu
End;
Procedure Set_Grid_Spacing;
Var
    TempSpacing: Integer;
    Finished: Boolean;
    OldColour: ScreenColour;
Begin
    UnitStatus (System, Parameter, Init);
    Install (US_F01, '1 dots');
    Install (US_F02, '2 dots');
    Install (US_F03, '4 dots');
    Install (US_F04, '5 dots');
    Install (US_F05, '10 dots');
    Install (US_F06, '20 dots');
    Install (US_F07, '50 dots');
    Install (US_F08, '100 dots');
    UnitStatus (System, Parameter, TurnOn);
    UnitClear (System);
    Repeat
        Finished := True;
        Case Decode Of
            US_F01:
                TempSpacing := 1;
            US_F02:
                TempSpacing := 2;
            US_F03:
                TempSpacing := 4;
            US_F04:

```

```

        TempSpacing := 5;
    US_F05:
        TempSpacing := 10;
    US_F06:
        TempSpacing := 20;
    US_F07:
        TempSpacing := 50;
    US_F08:
        TempSpacing := 100;
    Otherwise:
        Begin
            Ring_Bell;
            Finished := False
        End
    End
    {EndCase}
Until Finished;
If Grid_On
Then
    Begin
        Off_Grid; {Get rid of old grid}
        GridSpacing := TempSpacing * LambdaToGamma;
        On_Grid {Draw new grid}
    End
Else
    GridSpacing := LambdaToGamma * TempSpacing;
{EndIf}
Set_Menu
End;
Procedure Set_Current;
Var
    Finished: Boolean;
Begin
    UnitStatus (System, Parameter, Init);
    Install (US_F01, p_well);
    Install (US_F02, devwell);
    Install (US_F06, poly);
    Install (US_F05, polycap);
    Install (US_F03, p_guard);
    Install (SH_F01, glass);
    Install (US_F04, n_guard);
    Install (US_F07, n+);
    Install (US_F08, p+);
    Install (US_F09, contact);
    Install (US_F10, metal);
    UnitStatus (System, Parameter, TurnOn);
    UnitClear (System);
    Repeat
        Finished := True;
        Case Decode Of
            US_F01:
                LayerCurrent := PW;
            US_F02:
                LayerCurrent := F;
            US_F06:
                LayerCurrent := P;
            US_F05:
                LayerCurrent := CP;
            US_F03:
                LayerCurrent := PG;
            SH_F01:
                LayerCurrent := G;
            US_F04:
                LayerCurrent := NG;
            US_F07:
                LayerCurrent := NP;
            US_F08:
                LayerCurrent := PP;
            US_F09:
                LayerCurrent := C;

```

```

Finished := Eof (In File);
While Not Finished Do
Begin
  Case In_File Of
    'B':
      Begin
        Get (In File);
        Read (In_File, X_Size, Y_Size,
              X_Center, Y_Center);
        If Flip
        Then
          Swap (X_Size, Y_Size);
        {EndIf}
        Temp_X := Trunc (X_Center * t11 +
                        Y_Center * t21 + txx) -
                  X_Left;
        Temp_Y := Trunc (Y_Center * t12 +
                        X_Center * t22 + tyy) -
                  Y_Bottom;
        Writeln (Out_File, 'B', X_Size: 5,
                  Y_Size: 5, Temp_X: 5, Temp_Y: 5,
                  );
      End;
    'D':
      Begin
        Read (In File, Buffer);
        Writeln (Out_File, Buffer);
        Finished := True;
      End;
    Otherwise:
      Begin
        Read (In File, Buffer);
        Writeln (Out_File, Buffer);
      End;
  End;
{EndCase}
Readln (In File);
Finished := Finished Or Eof (In_File);
End
{EndWhile}
Else
  Writeln ('*** ', In_Name, ' is not a CIF file. ***');
{EndIf}
End;
Begin
  X_L := 9600;
  X_R := 0;
  Y_B := 9600;
  Y_T := 0;
  X_Left := 9600;
  Y_Bottom := 9600;
  Flip := False;
  Writeln;
  Writeln ('Program Name, ' - Version ', Program_Version,
           - Last_Modified: ', Program_Date);
  Writeln;
  Write ('File to read CIF symbol from: ');
  Readln (In_Name);
  Write ('File to write CIF symbol to: ');
  Readln (Out_Name);
  Finished := False;
  t11 := 1.0;
  t12 := 0.0;
  t21 := 0.0;
  t22 := 1.0;
  txx := 0.0;
  tyy := 0.0;
  Repeat
    Write ('Transformation (mirrotX mirrorY Rotate Help End): ');

```

```

        LayersVisible := LayersVisible + 'G';
CUS F01:
        LayersVisible := LayersVisible - 'PW';
CUS F02:
        LayersVisible := LayersVisible - 'F';
CUS F03:
        LayersVisible := LayersVisible - 'PG';
CUS F04:
        LayersVisible := LayersVisible - 'NG';
CUS F05:
        LayersVisible := LayersVisible - 'GP';
CUS F06:
        LayersVisible := LayersVisible - 'P';
CUS F07:
        LayersVisible := LayersVisible - 'NP';
CUS F08:
        LayersVisible := LayersVisible - 'PP';
CUS F09:
        LayersVisible := LayersVisible - 'C';
CUS F10:
        LayersVisible := LayersVisible - 'M';
CSH F01:
        LayersVisible := LayersVisible - 'G';
SH F10:
        Finished := True;
    Otherwise:
        Ring_Bell;
    End;
{EndCase}
    StatusContext
Until Finished;
    New_Screen;
    Set_Menu;
End;
{SSSeg_05}
Procedure Zero;
Var
    X_Ptr:      Xcompartments;
    Y_Ptr:      Ycompartments;
    Layer:      TypeLayer;
    Ptr_Temp:   pCompRect;
    TempLayer:  TypeLayer;
    TempVisible: Set Of TypeLayer;
    Deletions:  Integer;
Begin
    TempLayer := LayerCurrent;
    TempVisible := LayersVisible;
    LayersVisible := [];
    StatusContext;
    Set_Visibility;
    For Layer := PW To G Do
        Begin
            LayerCurrent := Layer;
            StatusContext;
            Deletions := 0;
            For X_Ptr := 0 To X_Compartments Do
                For Y_Ptr := 0 To Y_Compartments Do
                    If Compartments [X_Ptr, Y_Ptr] <> Nil
                    Then
                        With Compartments [X_Ptr, Y_Ptr].RectDescriptor
                        Do
                            If (Layer In Residents) And (Layer In LayersVisible)
                            Then
                                Begin
                                    Deletions := Deletions +
                                        Headers [Layer].NoOfEntries;
                                    Ptr_Temp := Headers [Layer].FirstRect;
                                    While Ptr_Temp.NextBlock <> Nil Do
                                        Ptr_Temp := Ptr_Temp.NextBlock;
                                    {EndWhile}
                                End
                            End
                        End
                    End
                End
            End
        End
    End

```

```

Ptr Temp^.NextBlock := Free Comp;
Free Comp := Headers [Layer].FirstRect;
Headers [Layer].FirstRect := Nil;
Headers [Layer].NoOfEntries := 0;
Residents := Residents - [Layer]

End
(EndIf)
(EndFor)
(EndWith)
(EndIf)
(EndFor)
End;
(EndFor)
LayerCurrent := TempLayer;
LayersVisible := TempVisible;
New Screen;
Status
End;
Procedure Slew_To;
Begin
  MarkHairs (X_Old, Y_Old);
  MarkHairs (CursorX, CursorY);
  X_Old := CursorX;
  Y_Old := CursorY;
  StatusCursor
End;
Procedure Mark_To;
Var
  X_Center:      TypeCoord;
  Y_Center:      TypeCoord;
  X_Size:        TypeMeasure;
  Y_Size:        TypeMeasure;
Begin
  If (CursorX <> X_Old) And (CursorY <> Y_Old)
  Then
    Begin
      X_Center := (CursorX + X_Old) Div 2;
      Y_Center := (CursorY + Y_Old) Div 2;
      X_Size := Abs (CursorX - X_Old);
      Y_Size := Abs (CursorY - Y_Old);
      SlewHairs (CursorX, CursorY);
      Draw Rectangle (X_Center, Y_Center, X_Size, Y_Size);
      SlewHairs (CursorX, CursorY);
      If Insert Object (LayerCurrent, X_Center, Y_Center,
        X_Size, Y_Size)
      Then
        (EndIf)
      End
    End
  Else
    Ring Bell
  (EndIf)
End; (Mark To)
Procedure Delete;
Var
  X_Center:      TypeCoord;
  Y_Center:      TypeCoord;
  X_Size:        TypeMeasure;
  Y_Size:        TypeMeasure;
Begin
  If (CursorX <> X_Old) And (CursorY <> Y_Old)
  Then
    Begin
      X_Center := (CursorX + X_Old) Div 2;
      Y_Center := (CursorY + Y_Old) Div 2;
      X_Size := Abs (CursorX - X_Old);
      Y_Size := Abs (CursorY - Y_Old);
      If Delete Object (LayerCurrent, X_Center, Y_Center,
        X_Size, Y_Size)
      Then
        Begin

```

```

        X_Size := X_Size Div 2;
        Y_Size := Y_Size Div 2;
        StewAbs (X_Center - X_Size, Y_Center - Y_Size);
        DrawAbs (X_Center + X_Size, Y_Center + Y_Size);
        SlewAbs (X_Center - X_Size, Y_Center + Y_Size);
        DrawAbs (X_Center + X_Size, Y_Center - Y_Size);
    End
Else
    Ring_Bell
{EndIf}
End
Else
    Ring_Bell
{EndIf}
End; {Delete}
Procedure SetStyle (Colour: ScreenColour);
Begin
    SetColour (Colour);
    New_Screen
End;
Procedure ListDirectory;
Type
    String8      =      String [8];
    pString8     =      ^String8;
Var
    pDefault:    pString8;
    Default:     String8;
    Volume:      VID;
    d:           Directory;
    BlkFlg:      Boolean;
    ValFlg:      Boolean;
    DevNbr:      Integer;
    I:           Integer;
    First:       Integer;
    Last:        Integer;
Function pOSdirNam: pString8; External;
Begin {DirList}
    CCdirIOinit;
    pDefault := pOSdirNam;
    Default := pDefault;
    Default := Copy (Default, 2, Length (Default) - 1);
    Erase_Screen;
    Select Command Window;
    Normal_Video;
    Erase_Screen;
    Curse_On;
    Write('Enter volume name (default is ', Default, '): ');
    ReadLn (Volume);
    Curse_Off;
    Erase_Screen;
    Select Root Window;
    If Length (Volume) = 0
    Then
        Volume := Default;
    {EndIf}
    GetVolDir (Volume, d, BlkFlg, DevNbr, ValFlg);
    If Not ValFlg
    Then
        Writeln ('*** Invalid directory ', Volume, ' ***')
    Else
        Begin
            With d[0] Do
                Begin
                    First := NextBlock;
                    Last := deovblock;
                    Writeln ('Volume: ', dvid, ', size = ', last: 1,
                        ' blocks, ', dnumfiles: 1, ' files');
                End;
            {EndWith}
            For I := 1 To d[0].dnumfiles Do

```

```

with d[i] Do
  Begin
    If First = FirstBlock
    Then
      Writeln ('...', First: 4,
        '...', FirstBlock - First: 4);
    {EndIf}
    Writeln (dtid, Copy ('', 1,
      17 - Length (dtid)),
      FirstBlock: 4, '...', NextBlock -
      FirstBlock: 4, daccess.day: 4, '...',
      Copy ('JanFebMarAprMayJunJulAugSepOctNovDec',
        3 * (daccess.month - 1) + 1, 3),
        '...', daccess.year: 2);
    First := NextBlock
  End;
EndWith;
EndFor;
If First = Last
Then
  Writeln ('...', First: 4, '...', Last
    - First: 4);
EndIf;
End;
{EndIf}
Cursor On;
Write ('Carriage return to continue');
Readln;
Cursor Off;
New Screen;
Status
End; {ListDirectory}
Procedure Script;
{This is the command dispatcher}
{It accepts tokens from the decode function, and executes}
{the corresponding procedure if there is one.}
Var
  Fed_Up: Boolean;
Begin
  Set Menu;
  Fed_Up := False;
  Repeat
    Case Decode Of
      CSR_UP:
        CursorUp;
      CSR_DOWN:
        CursorDown;
      CSR_RIGHT:
        CursorRight;
      CSR_LEFT:
        CursorLeft;
      US_F01:
        Mark To;
      SH_F01:
        Delete;
      US_F02:
        Efficiency;
      US_F03:
        New Screen;
      US_F05, SH_F05:
        Set Current;
      US_F06:
        On Grid;
      US_F07:
        Set Location;
      SH_F06:
        Off Grid;
      US_F08:
        Slow Slew;
      US_F09:

```

```

        Fast_Slew;
US_F10:-
    Slew_To;
CUS_F01, CSH_F01:
    Set_Visibility;
CSH_F02:
    Zero;
CUS_F02:
    Plot;
CUS_F03, CSH_F03:
    Set_Magnification;
CUS_F04, CSH_F04:
    Set_Slow_Slew;
CUS_F05, CSH_F05:
    Set_Fast_Slew;
CUS_F06, CSH_F06:
    Set_Jump_Screen;
CSH_F07:
    Set_Grid_Spacing;
CUS_F07:
    ListDirectory;
CUS_F08:
    Read_File;
CSH_F08:
    Write_File;
CUS_F10:-
    Begin
        If_Dirty
        Then
            Write_File;
        {EndIf}
        Fed_Up := Saved Or Not Dirty
    End;
CSH_F09:
    SetStyle (White);
CUS_F09:
    SetStyle (Flip);
CUS_F10:
    Dump_Space;
    Otherwise:
        Ring_Bell;
    End
{EndCase}
Until Fed_Up;
End;
Procedure Epilogue;
Begin {Epilogue}
    Select Command_Window;
    Normal_Video;
    Erase_Screen;
    Curse_On;
    Select Root_Window;
    Erase_Screen;
    TermGraphics
End; {Epilogue}
Procedure Lock_5;
Begin
    Prologue;
    Script;
    Epilogue
End;
{$$}
Begin
    Lock_1
End.

```


Appendix B-3 - CifBias Program

```

Program CifBias;
Const
  Program Name      = 'CifBias';
  Program Version   = '0.2';
  Program Date      = '1984 07 24';
Var
  In Name:          String [30];
  In File:          Interactive;
  Out Name:         String [30];
  Out File:         Interactive;
  Buffer:           String [80];
  X Left:          Integer;
  Y Bottom:        Integer;
  Cmd:             Char;
  r:               Real;
  rx:              Real;
  ry:              Real;
  st11:            Real;
  st12:            Real;
  st21:            Real;
  st22:            Real;
  stx:             Real;
  sty:            Real;
  t11:            Real;
  t12:            Real;
  t21:            Real;
  t22:            Real;
  txx:            Real;
  tyy:            Real;
  ptx:            Real;
  pty:            Real;
  X L:            Integer;
  X R:            Integer;
  Y B:            Integer;
  Y T:            Integer;
  Temp X:         Integer;
  Temp Y:         Integer;
  degree:         Integer;
  Flip:           Boolean;
  Okay:           Boolean;
  Finished:       Boolean;
Procedure Swap (Var Arg_1, Arg_2: Integer);
Var
  Temp:           Integer;
Begin
  Temp := Arg_1;
  Arg_1 := Arg_2;
  Arg_2 := Temp;
End;
Procedure Read_File;
Var
  X Center:       Integer;
  Y Center:       Integer;
  X Size:         Integer;
  Y Size:         Integer;
  X Dir:          Integer;
  Y Dir:          Integer;
  Finished:       Boolean;
Begin
  Readln (In File, Buffer);
  If 0 <> Pos ('DS', Buffer)
  Then
    Begin
      Finished := Eof (In File);
      While Not Finished Do
        Begin

```

```

Case In_File Of
  'B':
    Begin
      Get (In_File);
      Read (In_File, X_Size, Y_Size,
            X_Center, Y_Center);
      If X_Center - X_Size Div 2 < X_L
      Then
        X_L := X_Center - X_Size Div 2;
      {EndIf}
      If X_Center + X_Size Div 2 > X_R
      Then
        X_R := X_Center + X_Size Div 2;
      {EndIf}
      If Y_Center - Y_Size Div 2 < Y_B
      Then
        Y_B := Y_Center - Y_Size Div 2;
      {EndIf}
      If Y_Center + Y_Size Div 2 > Y_T
      Then
        Y_T := Y_Center + Y_Size Div 2;
      {EndIf}
      If Flip
      Then
        Swap (X_Size, Y_Size);
      {EndIf}
      Temp_X := Trunc (X_Center * t11 +
                      Y_Center * t21 + txx);
      Temp_Y := Trunc (X_Center * t12 +
                      Y_Center * t22 + tyx);
      If Temp_X - X_Size Div 2 < X_Left
      Then
        X_Left := Temp_X - X_Size Div 2;
      {EndIf}
      If Temp_Y - Y_Size Div 2 < Y_Bottom
      Then
        Y_Bottom := Temp_Y - Y_Size Div 2;
      {EndIf}
    End;
  'D':
    Finished := True;
  Otherwise:
    ; {Just in case.}
  End;
{EndCase}
ReadLn (In_File);
Finished := Finished Or Eof (In_File)
End
{EndWhile}
End
Else
  Writeln ('*** ', In_Name, ' is not a CIF file. ***');
{EndIf}
End;
Procedure Write_File;
Var
  X_Center: Integer;
  Y_Center: Integer;
  X_Size: Integer;
  Y_Size: Integer;
  X_Dir: Integer;
  Y_Dir: Integer;
  Finished: Boolean;
Begin
  ReadLn (In_File, Buffer);
  If 0 <> Pos ('DS', Buffer)
  Then
    Begin
      Writeln (Out_File, Buffer);
      Get (In_File);
    End
  End
End

```

```

Finished := Eof (In File);
While Not Finished Do
  Begin
    Case In File Of
      'B':
        Begin
          Get (In File);
          Read (In File, X_Size, Y_Size,
                X_Center, Y_Center);
          If Flip
            Then
              Swap (X_Size, Y_Size);
            {EndIf}
          Temp_X := Trunc (X_Center * t11 +
                          Y_Center * t21 + txx) -
                      X_Left;
          Temp_Y := Trunc (Y_Center * t12 +
                          Y_Center * t22 + tyy) -
                      Y_Bottom;
          Writeln (Out File, 'B', X_Size: 5,
                  Y_Size: 5, Temp_X: 5, Temp_Y: 5,
                  ');
        End;
      'D':
        Begin
          Read (In File, Buffer);
          Writeln (Out File, Buffer);
          Finished := True;
        End;
      Otherwise:
        Begin
          Read (In File, Buffer);
          Writeln (Out File, Buffer);
        End;
      End;
    {EndCase}
    Readln (In File);
    Finished := Finished Or Eof (In File);
  End
{EndWhile}
Else
  Writeln ('*** ', In_Name, ' is not a CIF file. ***');
{EndIf}
End;
Begin
  X_L := 9600;
  X_R := 0;
  Y_B := 9600;
  Y_T := 0;
  X_Left := 9600;
  Y_Bottom := 9600;
  Flip := False;
  Writeln;
  Writeln ('Program Name, ' - Version ', Program Version,
          ' - Last Modified: ', Program_Date);
  Writeln;
  Write ('File to read CIF symbol from: ');
  Readln (In_Name);
  Write ('File to write CIF symbol to: ');
  Readln (Out_Name);
  Finished := False;
  t11 := 1.0;
  t12 := 0.0;
  t21 := 0.0;
  t22 := 1.0;
  txx := 0.0;
  tyy := 0.0;
  Repeat
    Write ('Transformation (mirrotX mirrorY Rotate Help End): ');
  
```

```

Readln (Cmd);
Case Cmd Of
  'Y', 'y':
    Begin
      Writeln ('Mirrored across y axis. ');
      t11 := -t11;
      txx := -txx;
      t21 := -t21
    End;
  'X', 'x':
    Begin
      Writeln ('Mirrored across x axis. ');
      t22 := -t22;
      tyy := -tyy;
      t12 := -t12
    End;
  'R', 'r':
    Begin
      Write ('rotation (0, 90, 180, 270): ');
      Readln (degree);
      Okay := True;
      Case degree Of
        0:
          Okay := False;
        90:
          Begin
            Flip := Not Flip;
            rx := 0.0;
            ry := 1.0
          End;
        180:
          Begin
            rx := -1.0;
            ry := 0.0
          End;
        270:
          Begin
            Flip := Not Flip;
            rx := 0.0;
            ry := -1.0
          End;
        Otherwise:
          Begin
            Writeln ('Invalid response.', Chr (7));
            Okay := False
          End
      End;
    End;
  {EndCase}
  If Okay
  Then
    Begin
      st11 := t11 * rx - t12 * ry;
      st12 := t11 * ry + t12 * rx;
      st21 := t21 * rx - t22 * ry;
      st22 := t21 * ry + t22 * rx;
      stx := txx * rx - tyy * ry;
      sty := txx * ry + tyy * rx;
      t11 := st11;
      t12 := st12;
      t21 := st21;
      t22 := st22;
      txx := stx;
      tyy := sty
    End
  {EndIf}
  End;
  'E', 'e', ' ':
    Finished := True;
  Otherwise:
    Begin

```

```

        Writeln;
        Writeln ('X - mirror symbol across x axis. ');
        Writeln ('Y - mirror symbol across y axis. ');
        Writeln ('R - rotate symbol by specified angle. ');
        Writeln ('E - terminate transform specifications. ');
        Writeln ('Transformations are performed in the ');
        Writeln ('order specified. As many transformations ');
        Writeln ('as required may be specified. ');
        Writeln;
    End;
    {EndCase}
End
Until Finished;
If (Length (In_Name) <> 0) And (Length (Out_Name) <> 0)
Then
    Begin
        {SI-}
        Reset (In_File, In_Name);
        {SI+}
        If IoResult <> 0
        Then
            Begin
                Writeln ('*** Unable to open ', In_Name, ' ***');
                Exit (Cif_Bias);
            End;
        {EndIf}
        {SI-}
        Rewrite (Out_File, Out_Name);
        {SI+}
        If IoResult <> 0
        Then
            Begin
                Writeln ('*** Unable to open ', Out_Name, ' ***');
                Exit (Cif_Bias);
            End;
        {EndIf}
        X_Left := 9600;
        Y_Bottom := 9600;
        Read File;
        {Rewind input file.}
        Close (In_File, Lock);
        Reset (In_File, In_Name);
        Write File;
        Close (Out_File, Lock);
        Close (In_File, Lock);
        Writeln ('(X centers biased left by ', X_Left Div 2: 1,
            ' units.) ');
        Writeln ('(Y centers biased down by ', Y_Bottom Div 2: 1,
            ' units.) ');
        Writeln ('X Min = ', X_L Div 2:4, ' X Max = ', X_R Div 2:4);
        Writeln ('Y Min = ', Y_B Div 2:4, ' Y Max = ', Y_T Div 2:4);
        Writeln ('Finished. ');
    End
Else
    Writeln ('(Null response.) ')
{EndIf}
End.

```

```

Subroutine SetPrm (Cmd)
C-----C
C
Integer      Cmd
C
Integer      Stage, Stages, A(10,3)
Real          H(10), WZ(10), OZ(10), WP(10)
Real          QP(10), LoFreq, HiFreq, TwoPie
Common       /Info/ A, H, WZ, OZ, WP, QP, Stage,
                  Stages, LoFreq, HiFreq, TwoPie
C
Real          CrapR
Integer       CrapI
Junk = WnSys (2)
Write (Unit = *, Fmt = '(A, A) Char (27), "J"
Goto (110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220)
C
110  Continue
      Call GetInt ('Current value of k1 is: ', A(Stage,1))
      A(Stage,1) = Min (1, Max (A(Stage,1), -1))
      Goto 500
120  Continue
      Call GetInt ('Current value of k2 is: ', A(Stage,2))
      A(Stage,2) = Min (1, Max (A(Stage,2), -1))
      Goto 500
130  Continue
      Call GetInt ('Current value of k3 is: ', A(Stage,3))
      A(Stage,3) = Min (1, Max (A(Stage,3), -1))
      Goto 500
140  Continue
      RTemp = WZ(Stage) / TwoPie
      Call GetRel ('Current value of Woz in hertz is: ', RTemp)
      WZ(Stage) = RTemp * TwoPie
      Goto 500
150  Continue
      Call GetRel ('Current value of Qz is: ', QZ(Stage))
      Goto 500
160  Continue
      RTemp = WP(Stage) / TwoPie
      Call GetRel ('Current value of Wop in hertz is: ', RTemp)
      WP(Stage) = RTemp * TwoPie
      Goto 500
170  Continue
      Call GetRel ('Current value of Qp is: ', QP(Stage))
      Goto 500
180  Continue
      Call GetInt ('Currently selected stage is: ', Stage)
      Stage = Max (1, Min (Stage, 10))
      Call Part2
      Goto 500
190  Continue
      Call GetInt ('Active number of stages is: ', Stages)
      Stages = Max (1, Min (Stages, 10))
      Goto 500
200  Continue
      Call GetRel ('Current low frequency in hertz is: ', LoFreq)
      Goto 500
210  Continue
      Call GetRel ('Current high frequency in hertz is: ', HiFreq)
      Goto 500
220  Continue
      Call GetRel ('Current gain factor is: ', H(Stage))
      Goto 500
500  Continue
      Junk = WnSys (1)
      End
C-----C

```

Appendix C-1 - Complex Program

```

Ssegment Grot
Program
CAL
C
Integer      Stage, Stages, A(10,3)
Real         H(10), WZ(10), QZ(10), WP(10)
Real         QP(10), LoFreq, HiFreq, TwoPie
Common       /Info/  A, H, WZ, QZ, WP, QP, Stage,
                  Stages, LoFreq, HiFreq, TwoPie
C
Integer      Locs(8)
Character*8  Lbls(8)
Common       /Label1/ Locs
Common       /Label2/ Lbls
C
Real         Freq(200), Mag(200), Phase(200)
Common       /Info2/ Freq, Mag, Phase
C
Integer      Cmd
C
Call Instal
20 Continue
    Call Status
    Call GetHit (Cmd)
    Goto (100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
        : 110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
        : 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
        : 130, 131, 132, 133, 134, 135, 136, 137, 138, 139) Cmd + 1
    Goto 10
100 Continue
C      Set parameter - current stage
    Call SetPrm (8)
    Goto 10
101 Continue
    Goto 10
102 Continue
C      Set parameter - gain factor
    Call SetPrm (12)
    Goto 10
103 Continue
C      Set parameter - Wp
    Call SetPrm (6)
    Goto 10
104 Continue
C      Set parameter - Qp
    Call SetPrm (7)
    Goto 10
105 Continue
C      Set parameter - lofreq
    Call SetPrm (10)
    Goto 10
106 Continue
C      Set parameter - hifreq
    Call SetPrm (11)
    Goto 10
107 Continue
C      Display coefficients
    Call DispCf
    Goto 10
108 Continue
C      Concept plot
    Call PltCon
    Goto 10
109 Continue
    Goto 10
110 Continue
C      Set parameter - kl

```

```
      Call SetPrm (1)
      Goto 10
111  Continue
   C   Set parameter - k2
      Call SetPrm (2)
      Goto 10
112  Continue
   C   Set parameter - k3,
      Call SetPrm (3)
      Goto 10
113  Continue
   C   Set parameter - Wz
      Call SetPrm (4)
      Goto 10
114  Continue
   C   Set parameter - Qz
      Call SetPrm (5)
      Goto 10
115  Continue
   C   Set parameter - lofreq
      Call SetPrm (10)
      Goto 10
116  Continue
   C   Set parameter - hifreq
      Call SetPrm (11)
      Goto 10
117  Continue
   C   Display coefficients
      Call DispCf
      Goto 10
118  Continue
   C   Concept plot
      Call PltCon
      Goto 10
119  Continue
   C   Save file and exit
      Call SvFile
      Goto 30
      Goto 10
120  Continue
   C   Set parameter - number of stages
      Call SetPrm (9)
      Goto 10
121  Continue
      Goto 10
122  Continue
      Goto 10
123  Continue
   C   Write linear response file.
      Call PrResp (0)
      Goto 10
124  Continue
   C   Write log response file.
      Call PrResp (1)
      Goto 10
125  Continue
      Goto 10
126  Continue
   C   LA120 plot
      Call Plt120
      Goto 10
127  Continue
   C   HP7221 plot
      Call PltHP
      Goto 10
128  Continue
   C   Concept Plot
      Call PltCon
      Goto 10
129  Continue
```



```
C Subroutine SetPrm (Cmd) C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C Integer Cmd
C Integer Stage, Stages, A(10,3)
Real H(10), WZ(10), QZ(10), WP(10)
Real QP(10), LoFreq, HiFreq, TwoPie
Common Info/ A, H, WZ, QZ, WP, QP, Stage,
Stages, LoFreq, HiFreq, TwoPie
C Real CrapR
Integer CrapI
Junk = WnSys (2)
Write (Unit = *, Fmt = '(A,A)' Char (27), 'J'
Goto (110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220)
C 110 Continue
Call GetInt ('Current value of k1 is: ', A(Stage,1))
A(Stage,1) = Min (1, Max (A(Stage,1), -1))
Goto 500
C 120 Continue
Call GetInt ('Current value of k2 is: ', A(Stage,2))
A(Stage,2) = Min (1, Max (A(Stage,2), -1))
Goto 500
C 130 Continue
Call GetInt ('Current value of k3 is: ', A(Stage,3))
A(Stage,3) = Min (1, Max (A(Stage,3), -1))
Goto 500
C 140 Continue
RTemp = WZ(Stage) / TwoPie
Call GetRel ('Current value of Woz in hertz is: ', RTemp)
WZ(Stage) = RTemp * TwoPie
Goto 500
C 150 Continue
Call GetRel ('Current value of Qz is: ', QZ(Stage))
Goto 500
C 160 Continue
RTemp = WP(Stage) / TwoPie
Call GetRel ('Current value of Wop in hertz is: ', RTemp)
WP(Stage) = RTemp * TwoPie
Goto 500
C 170 Continue
Call GetRel ('Current value of Qp is: ', QP(Stage))
Goto 500
C 180 Continue
Call GetInt ('Currently selected stage is: ', Stage)
Stage = Max (1, Min (Stage, 10))
Call Part2
Goto 500
C 190 Continue
Call GetInt ('Active number of stages is: ', Stages)
Stages = Max (1, Min (Stages, 10))
Goto 500
C 200 Continue
Call GetRel ('Current low frequency in hertz is: ', LoFreq)
Goto 500
C 210 Continue
Call GetRel ('Current high frequency in hertz is: ', HiFreq)
Goto 500
C 220 Continue
Call GetRel ('Current gain factor is: ', H(Stage))
Goto 500
C 500 Continue
Junk = WnSys (1)
End
```

```

C      Subroutine Eval
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Integer      Stage, Stages, A(10,3)
C      Real          H(10), WZ(10), OZ(10), WP(10)
C      Real          QP(10), LoFreq, HiFreq, TwoPie
C      Common        /Info/ A, H, WZ, OZ, WP, QP, Stage,
C                      Stages, LoFreq, HiFreq, TwoPie
C
C      Integer      Count
C      Real          G(10), J(10,3), K(10,3)
C      Common        /Info3/ G, J, K, Count
C
C      Real          Freq(200), Mag(200), Phase(200)
C      Common        /Info2/ Freq, Mag, Phase
C
C      Real          DeltaF
C      Real          Flo
C      Real          Phi
C      Logical       Linear
C      Linear = .False.
C      Goto 10
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Entry EvalLn
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Linear = .True.
10      Continue
C      Count = Stages
C      Do 20 I = 1, Stages
C          G(I) = H(I)
C          J(I, 1) = A(I, 3) * WZ(I) * WZ(I)
C          J(I, 2) = A(I, 2) * WZ(I) / OZ(I)
C          J(I, 3) = A(I, 1)
C          K(I, 1) = WP(I) * WP(I)
C          K(I, 2) = WP(I) / QP(I)
C          K(I, 3) = 1.0
20      Continue
C      If (Linear)
C      .Then
C          DeltaF = (HiFreq - LoFreq) / 199.0
C          Do 30 I = 1, 200
C              Freq(I) = LoFreq + (I - 1) * DeltaF
C              Call MagPhs (Freq(I) * TwoPie, Mag(I), Phase(I))
C              Phase(I) = Phase(I) * 360.0 / TwoPie
30      Continue
C      Else
C          Flo = Alog10 (LoFreq)
C          Phi = Alog10 (HiFreq)
C          DeltaF = (Phi - Flo) / 199.0
C          Do 40 I = 1, 200
C              Freq(I) = 10.0 ** (Flo + (I - 1) * DeltaF)
C              Call MagPhs (Freq(I) * TwoPie, Mag(I), Phase(I))
C              Phase(I) = Phase(I) * 360.0 / TwoPie
40      Continue
C      EndIf
C      End
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Subroutine PltCon
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Real          Freq(200), Mag(200), Phase(200)
C      Common        /Info2/ Freq, Mag, Phase

```

```

C      Call Eval
      Call Plot (Freq, Mag, Phase, 200)
      End

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Subroutine PltHP
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Real      Freq(200), Mag(200), Phase(200)
      Common      /Info2/ Freq, Mag, Phase

C      Logical      Init, Junk
      Integer*1      Temp
      Temp = 1
      Call Term
      Junk = Init (Temp)
      Call Eval
      Call Plot (Freq, Mag, Phase, 200)
      Call Term
      Temp = 0
      Junk = Init (Temp)
      End

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Subroutine PltI20
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Real      Freq(200), Mag(200), Phase(200)
      Common      /Info2/ Freq, Mag, Phase

C      Logical      Init, Junk
      Integer*1      Temp
      Call Term
      Write (Unit = *, Fmt = '(A, A)' Char(27), 'J'
      Write (Unit = *, Fmt = 1000)
      Temp = 2
      Junk = Init (Temp)
      Call Eval
      Call Plot (Freq, Mag, Phase, 200)
      Call Term
      Temp = 0
      Junk = Init (Temp)
      Call Plot (Freq, Mag, Phase, 200)

1000  Format ('DecWriter plots take a LONG time, (approximately 5',
      .      'minutes), so dont'
      .      'dispair. The screen will be redrawn when the plot file',
      .      'is closed.'
      .      'See a TA for instructions on how to get the plot.')

      End

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Subroutine PrResp (Mode)
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Real      Freq(200), Mag(200), Phase(200)
      Common      /Info2/ Freq, Mag, Phase

C      Open (Unit = 7, File = 'RESPONSE.TEXT', Status = 'New', Err = 99)
      If (Mode .eq. 1)
      .Then
          Call Eval

```

[illegible]

```

C                                     C
C      Subroutine Dump (IUNIT)
C      -----C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Integer      IUNIT
C
C      Integer      Stage, Stages, A(10,3)
C      Real          H(10), WZ(10), QZ(10), WP(10)
C      Real          QP(10), LoFreq, HiFreq, TwoPie
C      Common        /Info/ A, H, WZ, QZ, WP, QP, Stage,
C                        Stages, LoFreq, HiFreq, TwoPie
C
C      Write (Unit = IUNIT, Fmt = 2000)
C      Write (Unit = IUNIT, Fmt = 1000)
C      Write (Unit = IUNIT, Fmt = 2000)
C      Do 10 I = 1, 10
C          Write (Unit = IUNIT, Fmt = 3000) I, A(I,1), A(I,2), A(I,3);
C                                          H(I), WZ(I) / TwoPie, QZ(I),
C                                          WP(I) / TwoPie, QP(I)
10      Continue
C      Write (Unit = IUNIT, Fmt = 2000)
C      Write (Unit = IUNIT, Fmt = 4000) Stages, LoFreq, HiFreq
C      Write (Unit = IUNIT, Fmt = 2000)
1000      Format (
2000      .- Stage | k1 | k2 | k3 | Ho | Woz (Hz) |
      .- Qz. | Wop (Hz) | Qp |
2000      Format (*
      .- +-----+
      .- +-----+
4000      Format (- Active stages: -, I2, -, Low frequency: -, F10.2,
3000      Format (-, 3X, I2, 2X, -, High frequency: -, F10.2,
      .- 5(1X, E12.5E2, 1X, -, )
      End
C
C      Subroutine MagPhs (Omega, Mag, Phase)
C      -----C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Real          Omega
C      Real          Mag
C      Real          Phase
C
C      Integer      Count
C      Real          G(10), J(10,3), K(10,3)
C      Common        /Info3/ G, J, K, Count
C
C      Integer      Stage
C      Complex      Result
C      Complex      Respon
C      Result = (1.0, 0.0)
C      Do 10 Stage = 1, Count
C          Result = Result * Respon (Stage, Omega)
10      Continue
C      Mag = 20.0 * ALog10 (CABS (Result))
C      Phase = ATan2 (AIMag (Result), Real (Result))
C      End
C
C      Complex Function Respon (Stage, Omega)
C      -----C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Integer      Stage
C      Real          Omega

```

```

Integer      Count
Real          G(10), J(10,3), K(10,3)
Common       /Info3/ G, J, K, Count

C
Complex      S
S = Cmplx (0.0, Omega)
Respon = G(Stage) *
.           (J(Stage,1) + S * (J(Stage,2) + S * J(Stage,3))) /
.           (K(Stage,1) + S * (K(Stage,2) + S * K(Stage,3)))
End

C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
Subroutine Instal
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
Logical      Temp, Init
Integer*4    Junk

C
Integer      Stage, Stages, A(10,3)
Real         H(10), WZ(10), QZ(10), WP(10)
Real         QP(10), LoFreq, HiFreq, TwoPie
Common       /Info/ A, H, WZ, QZ, WP, QP, Stage,
                  Stages, LoFreq, HiFreq, TwoPie

C
Commence with: 10 Stage
                using Ws and Qs

C
Stage = 1
Stages = 1

C
Call SegLck
Open (Unit = 2, File = '/SYSTEM', Buffered = 'UNBUFFERED')
Call WnInit

C
Initialize the Pascal GetHit routine.

C
Call InitIO

C
Initialize graphics system and erase screen.

C
Junk = 0
Temp = Init (Junk)
Write (Unit = *, Fmt = '(A, A)' Char (27), 'J'

C
Do 10 I = 1, 10
  A(I, 1) = 1
  A(I, 2) = 1
  A(I, 3) = 1
  H(I) = 1.0
  WZ(I) = 0.0
  QZ(I) = 1.0E6
  WP(I) = 0.0
  QP(I) = 1.0
10 Continue
Call Part1
TwoPie = 8.0 * Atan (1.0)
LoFreq = 1.0
HiFreq = 10000.0
End

C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
Subroutine Part1
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
Integer      Locs(8)
Character*8   Lbls(8)

```

```

Common      /Label1/ Locs
Common      /Label2/ Lbls

C
C
Call LbInit

C
Call Stuff (00, 'SelStage')
Call Stuff (15, 'Set Low')
Call Stuff (05, 'Frequenc')
Call Stuff (16, 'Set High')
Call Stuff (06, 'Frequenc')
Call Stuff (17, 'Display')
Call Stuff (07, 'Coeffs')
Call Stuff (18, 'Concept')
Call Stuff (08, 'Plot')
Call Stuff (19, 'Exit')
Call Stuff (30, 'Set # Of')
Call Stuff (20, 'Stages')
Call Stuff (33, 'WriteLin')
Call Stuff (23, 'Response')
Call Stuff (34, 'WriteLog')
Call Stuff (24, 'Response')
Call Stuff (36, 'LA120')
Call Stuff (26, 'Plot')
Call Stuff (37, 'HP7221')
Call Stuff (27, 'Plot')
Call Stuff (38, 'Concept')
Call Stuff (28, 'Plot')
Call Stuff (39, 'LoadFile')
Call Stuff (29, 'SaveFile')

C
Locs (1) = 10
Lbls (1) = 'k1 ( )'
Locs (2) = 11
Lbls (2) = 'k2 ( )'
Locs (3) = 12
Lbls (3) = 'k3 ( )'
Locs (4) = 13
Lbls (4) = 'Woz ( )'
Locs (5) = 14
Lbls (5) = 'Qz ( )'
Locs (6) = 03
Lbls (6) = 'Wop ( )'
Locs (7) = 04
Lbls (7) = 'Op ( )'
Locs (8) = 02
Lbls (8) = 'Ho ( )'
Call Part2

C
End

C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
Subroutine Part2
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
Character*2   Arg
C
Integer       Stage, Stages, A(10,3)
Real          H(10), WZ(10), QZ(10), WP(10)
Real          OP(10), LoFreq, HiFreq, TwoPie
Common        /Info/   A, H, WZ, QZ, WP, OP, Stage,
                      Stages, LoFreq, HiFreq, TwoPie
C
Integer       Locs(8)
Character*8    Lbls(8)
Common        /Label1/ Locs
Common        /Label2/ Lbls
C
Write (Unit = Arg, Fmt = '(12.2)') Stage

```



```

      Do 10 I = 1, 8
        Lbls(I)(5:6) = Arg
        Call Stuff (Locs(I), Lbls(I))
10    Continue
      Call LbOn
      End

C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      Subroutine Stuff (Key, Disp)
C-----C
C
C
      Integer      Key
      Character*8   Disp
      Character*16   Temp
      Temp = Char (27) // '-' // Char (Key + IChar (' '))
      Junk = LbSet (Key, Disp, Temp)
      End

```

Appendix C-2 - Plotting Support For Complex Program

```

{SR-}
Unit PlotData;
Interface
  Uses
    {Su /ccutil/graphlib.obj}
    GraphDefs,
    Graph;
  Type
    Type Real Array = Array [1..1000] Of Real;
  Procedure Plot
    (Freq, Mag, Phase: Type_Real_Array;
     Var Points: LongInt);
  Function Init
    (Var Device: DeviceType): Boolean;
  Procedure Term;
{EndInterface}
Implementation
  Function Init;
  Var
    X_Min: Integer;
    X_Max: Integer;
    Y_Min: Integer;
    Y_Max: Integer;
    X_Dots: Integer;
    Y_Dots: Integer;
  Begin
    Init := InitGraphics (Device);
    InqDevice (X_Min, X_Max, Y_Min, Y_Max, Y_Dots, Y_Dots);
    Case Device Of
      LA120:
        Begin
          X_Max := 10 * X_Dots;
          Y_Max := 8 * Y_Dots;
          SetDevice (X_Min, X_Max, Y_Min, Y_Max);
          SetColour (Black);
        End;
      HP7221:
        Begin
          X_Max := Round (10.0 * 25.4 * X_Dots);
          Y_Max := Round (8.0 * 25.4 * Y_Dots);
          SetDevice (X_Min, X_Max, Y_Min, Y_Max);
          SetColour (Black);
        End;
      Otherwise:
        SetColour (White)
    End;
  {EndCase}
End;
Procedure Term;
Begin
  TermGraphics
End;
Procedure Plot;
Const
  Magnitude Ticks = 41;
  Phase Ticks = 7;
  Bound = 1000.0;
  TextWidth = 6;
  TextHeight = 16;
  vp_y_max = 0.95;
  vp_y_min = 0.05;
  vp_x_min = 0.1;
  vp_x_max = 0.9;
  ticks_to_window = 18; {Must be divisible by 2 and 3.}
Var
  Temp: Integer;
  Y_Min: Integer;
  Y_Max: Integer;

```

```

Char Height:      Integer;
XHome:            Integer;
XSize:            Integer;
YHome:            Integer;
YSize:            Integer;
FreMax:           Real;
FreMin:           Real;
DeltaF:           Real;
MagMax:           Real;
MagMin:           Real;
DeltaM:           Real;
X:                Integer;
DeltaT:           Real;
FirstT:           Real;
I:                Integer;
Result:           String255;
dF:               Real;
FS:               Real;
FE:               Real;
Slope:            Real;
Intercept:        Real;
Ticks:            Integer;
Function Log (Arg: Real): Real;
Begin
    Log := Ln (Arg) / Ln (10)
End;
Function MaxReal (Arg_1, Arg_2: Real): Real;
Begin
    If Arg_1 > Arg_2
    Then
        MaxReal := Arg_1
    Else
        MaxReal := Arg_2
    {EndIf}
End;
Function MinReal (Arg_1, Arg_2: Real): Real;
Begin
    If Arg_1 < Arg_2
    Then
        MinReal := Arg_1
    Else
        MinReal := Arg_2
    {EndIf}
End;
Function ModReal (Arg_1, Arg_2: Real): Real;
Begin
    ModReal := Arg_1 - Arg_2 * Trunc (Arg_1 / Arg_2)
End;
Function SignReal (Arg: Real): Real;
Begin
    If Arg < 0.0
    Then
        SignReal := -1.0
    Else
        SignReal := +1.0
    {EndIf}
End;
Function CeilReal (Arg: Real): Real;
Begin
    If Trunc (Arg) = Arg
    Then
        CeilReal := Arg
    Else
        CeilReal := Trunc (Arg + SignReal (Arg))
    {EndIf}
End;
Function FloorReal (Arg: Real): Real;
Begin
    If Trunc (Arg) = Arg
    Then

```

```

        FloorReal := Arg
    Else
        FloorReal := Trunc (Arg)
    {EndIf}
End;
Begin
    FreMax := Log (Freq[Points]);
    FreMin := Log (Freq[1]);
    DeltaF := FreMax - FreMin;
    MagMax := MinReal (Mag[1], +Bound);
    MagMin := MaxReal (Mag[1], -Bound);
    For I := 1 To Points Do
        Begin
            If Mag[I] > +Bound
            Then
                Mag[I] := +Bound
            Else
                If Mag[I] < -Bound
                Then
                    Mag[I] := -Bound;
                {EndIf}
            {EndIf}
            If Mag[I] > MagMax
            Then
                MagMax := Mag[I]
            Else
                If Mag[I] < MagMin
                Then
                    MagMin := Mag[I];
                {EndIf}
            {EndIf}
        End;
    {EndFor}
    If MagMin < 0.0
    Then
        MagMin := MaxReal (CeilReal (MagMin), -Bound)
    Else
        MagMin := FloorReal (MagMin);
    {EndIf}
    If MagMax >= 0.0
    Then
        MagMax := MinReal (CeilReal (MagMax), +Bound)
    Else
        MagMax := FloorReal (MagMax);
    {EndIf}
    Ticks := Magnitude Ticks;
    DeltaM := MagMax - MagMin;
    DeltaT := DeltaM / (Ticks - 1);
    NewPage;
    SetViewport (0.0, 1.0, 0.0, 1.0);
    SetWindow (0, 100, 0, 100);
    SlewAbs (0, 0);
    DrawAbs (100, 0);
    DrawAbs (100, 100);
    DrawAbs (0, 100);
    DrawAbs (0, 0);
    Char Height := (2 * ticks_to_window) Div 3;
    Y Min := 0;
    Y Max := (Ticks - 1) * ticks_to_window;
    Slope := (Y Max - Y Min) / (vp Y max - vp Y min);
    Intercept := 0.0 - Slope * vp Y Min;
    Y Min := Y Min - Char Height Div 2;
    Y Max := Y Max + Char Height Div 2;
    SetViewport (0.0, vp Y min, (Y Min - Intercept) Slope,
        (Y Max - Intercept) Slope);
    SetWindow (0, 42, Y Min, Y Max);
    For I := Magnitude Ticks - 1 DownTo 0 Do
        Begin
            Temp := I * Ticks To Window;
            Real_To_Str (MagMax = (Ticks - 1 - I) * DeltaT, 2,

```

```

        Result);
    If Length (Result) - Pos ('.', Result) < 2
    Then
        Result := Concat (Result, '0');
    {EndIf}
    SlewAbs (40, Temp);
    DrawAbs (42, Temp);
    TextAbs (39 - Length (Result) * TextWidth, Temp -
        Char Height Div 2, Result, 0.0, 90.0,
        TextWidth, Char_Height);
End;
{EndFor}
SetViewport (vp x max, 1.0, (Y Min - Intercept) / Slope,
    (Y Max - Intercept) / Slope);
For I := Phase_Ticks - 1 DownTo 0 Do
    Begin
        Temp := I * 120;
        Real_To_Str ((I - 3) * 60.0, 1, Result);
        SlewAbs (0, Temp);
        DrawAbs (2, Temp);
        Result := Concat (Copy ('', 1, 6 -
            Length (Result)), Result);
        TextAbs (4, Temp - Char Height Div 2, Result, 0.0,
            90.0, TextWidth, Char_Height);
    End;
{EndFor}
SetViewport (vp x min, vp x max, vp y min, vp y max);
Y Min := Y Min - Char Height Div 2;
Y Max := Y Max - Char Height Div 2;
SetWindow (0, 700, Y Min, Y Max);
SlewAbs (0, Y Min);
DrawAbs (700, Y Min);
DrawAbs (700, Y Max);
DrawAbs (0, Y Max);
DrawAbs (0, Y Min);
Fs := Freq[1] + ModReal (Freq[1], 100.0);
Fe := Freq[Points] - ModReal (Freq[Points], 100.0);
dF := PwrOfTen (Trunc (Log (Fs)));
While Fs <= Fe Do
    Begin
        X := Trunc ((Log (Fs) - FreMin) * 700 / DeltaF);
        SlewAbs (X, Y Min);
        DrawAbs (X, Y Max);
        Fs := Fs + dF;
        dF := PwrOfTen (Trunc (Log (Fs)));
    End;
{EndWhile}
SlewAbs (Trunc ((Log (Freq[1]) - FreMin) * 700 / DeltaF),
    Trunc ((Mag[1] - MagMin) *
        (Y Max - Y Min) / DeltaM));
For I := 2 To Points Do
    DrawAbs (Trunc ((Log (Freq[I]) - FreMin) * 700 / DeltaF),
        Trunc ((Mag[I] - MagMin) * (Y Max - Y Min) /
            DeltaM));
{EndFor}
MagMax := +180.0;
MagMin := -180.0;
DeltaM := 360.0;
SlewAbs (Trunc ((Log (Freq[1]) - FreMin) * 700 / DeltaF),
    Trunc ((Phase[1] - MagMin) * (Y Max - Y Min) /
        DeltaM));
For I := 2 To Points Do
    DrawAbs (Trunc ((Log (Freq[I]) - FreMin) * 700 / DeltaF),
        Trunc ((Phase[I] - MagMin) * (Y Max - Y Min) /
            DeltaM));
{EndFor}
SetViewport (0.0, 1.0, 0.0, vp y min);
SetWindow (0, 360, -10, 30);
Char Height := 12;
Real_To_Str (Freq[1], 2, Result);

```

```

TextAbs (Round (360 * vp * min) - TextWidth *
          Length (Result) Div 2, 0, Result, 0.0, 90.0,
          TextWidth, Char Height);
Real To Str (Freq[Points], 2, Result);
TextAbs (Round (360 * vp * max) - TextWidth *
          Length (Result) Div 2, 0, Result, 0.0, 90.0,
          TextWidth, Char Height);
Result := 'LOG (FREQUENCY)';
TextAbs ((360 - TextWidth * Length (Result)) Div 2, 0,
          Result, 0.0, 90.0, TextWidth, Char Height);
SetViewport (0.0, 1.0, vp_y max, 1.0);
TextAbs (6, 6, 'MAG (DB)', 0.0, 90.0,
          TextWidth, Char Height);
Result := 'PHASE';
TextAbs (345 - TextWidth * Length (Result), 6, Result, 0.0,
          90.0, TextWidth, Char Height);
End;
{EndImplementation}
End.

```

Appendix C-3 - Export Program

Program Export;

Program: Export

Purpose: Export is used to export a file from the concept environment to another host (in this case, either a Honeywell Level 66 running CP-6, or an Xerox Sigma running CP-5). The synchronization mechanism involves the reception of an ascii from the host, before a record is transmitted. The program checks the echo of each transmitted character in each transmitted record, to ensure that it actually got there. No retransmission facilities are provided, but the user is informed of an echoed error.

Usage: The following represents a typical session.
 (A picture is worth a thousand words!)
 From terminal mode, send the following to the host.
 !C ME TO CIF CARLETON 1
 Do not hit <CR> at the end of the line!
 Exit the terminal mode.
 Incant EXPORT
 The rest is self explanatory.

Author: DDRG @ C.U. Dept of Electronics

Date written: 83 04 15

Last modified: 84 04 19 - changed to datacom 1

Uses

```
{SU /CCUTIL/CCLIB}
CCclkiO;
```

Const

```
Asc NUL      = 0;
Asc CR       = 13;
Asc ESC      = 27;
Asc F        = 10;
Rx Timeout   = 30000;
Tx Timeout   = 5000;
DtComm1      = 31;
DtComm2      = 32;
```

Note: data comm 1 - \$30f4x
 2 - \$30f2x

```
Adr Data      = $30F21;
Adr Status    = $30F23;
Adr Command   = $30F25;
Adr Control   = $30F27;
```

Type

```
TypeByte      = -128..127;
TypeBits       = Packed Array [0..7] Of Boolean;
BitByte        = Record
    Case Boolean Of
        True:   (Bit: TypeBits);
        False:  (Byte: TypeByte);
    {EndCase}
End;
TypeAddr       = Record
    Case Boolean Of
        True:   (Value: TypeByte);
```

Carleton University

Department of Electronics

A Guide to Using the Corvus Concept

For Computer Aided Design

1984 09 13 - Revision 1.1

Duncan Glendinning and Miles Copeland


```

    Command.Address := ADR Command;
    Command.Value := Setup.Byte;
End; {Rem_Init}
Procedure Delay_Transmit (Arg: Char);
Const
    Wait_Time = 1667;
Var
    I: Integer;
Begin
    For I := 1 To Wait_Time Do;
        {EndFor}
    Rem_Transmit (Arg);
End;
Procedure Echo_Transmit (Arg: Char);
Const
    Tries = 20;
Var
    Count: Integer;
    Result: Char;
    Status: (Finished, Waiting, Error);
Begin
    Status := Waiting;
    Count := 0;
    Rem_Transmit (Arg);
    Repeat
        If Rem_Receive (Result)
        Then
            If Arg = Result
            Then
                Status := Finished
            Else
                Count := Count + 1;
            {EndIf}
        Else
            Count := Count + 1;
        {EndIf}
        If (Count > Tries) And (Status <> Finished)
        Then
            Status := Error;
        {EndIf}
    Until Status <> Waiting;
    If Status = Error
    Then
        Writeln ('*** Echo mismatch in <Echo_Transmit>. ***',
            Chr (7));
    {EndIf}
End;
Procedure Spit (Buffer: String255);
Var
    Ptr: Integer;
Begin {Spit}
    For Ptr := 1 To Length (Buffer) Do
        Echo_Transmit (Buffer [Ptr]);
    {EndFor}
    Echo_Transmit (Asc_CR);
End; {Spit}
Procedure Process;
Const
    Tries = 20;
Var
    Status: (Importing, Waiting, Error);
    Count: Integer;
    Buffer: String255;
    Temp: String255;
    Rec_Count: LongInt;
    Arg: Char;
    Start_Time: ClkStr40;
    Stop_Time: ClkStr40;
Begin {Process}
    Writeln; {I love symmetry.}

```

```

Rec_Count := 0;
Rem_Init;
ReadLn (In_File, Buffer);
Status := Importing;
ClkTime (Start_Time);
{Cause pending host command to be emitted...}
Delay_Transmit (Asc CR);
While (Status = Importing) And Not Eof (In_File) Do
Begin
    Count := 0;
    Status := Waiting;
    Repeat
        If Not Rem_Receive (Arg)
        Then
            Count = Count + 1;
        Else
            If Arg = ' '
            Then
                Count = Count + 1;
            Else
                Status := Importing;
            {EndIf}
        {EndIf}
        If (Count > Tries) And (Status <> Importing)
        Then
            Status := Error;
        {EndIf}
    Until Status <> Waiting;
    If Status = Importing
    Then
        Begin
            Rec_Count := Rec_Count + 1;
            WriteLn (Rec_Count, ' ', Buffer);
            Temp := Buffer;
            ReadLn (In_File, Buffer);
            SpIt (Temp)
        End
    {EndIf}
    End;
{EndWhile}
If Status = Error
Then
    WriteLn ('*** Failed to receive proceed
character from host. ***', Chr(7));
{EndIf}
Delay_Transmit (Asc ESC); {Signal end of file to pci}
Delay_Transmit (Asc F);
ClkTime (Stop_Time);
WriteLn;
WriteLn('Transmission summary - stopped at - ', Stop_Time);
WriteLn('started at - ', Start_Time);
WriteLn('file name - ', In_Name);
WriteLn('records - ', Rec_Count:1);
End; {Process}
Begin
    WriteLn;
    Write ('File to be exported: ');
    ReadLn (In_Name);
    {$I-}
    Reset (In_File, In_Name);
    {$I+}
    If IoResult <> 0
    Then
        WriteLn ('*** Error in opening ', In_Name, '. ***', Chr(7))
    Else
        Process
    {EndIf}
End.

```

Appendix D - Handbook As Seen By Students

Carleton University

Department of Electronics

A Guide to Using the Corvus Concept

For Computer Aided Design

1984 09 13 - Revision 1.1

Duncan Glendinning and Miles Copeland

no guarantee that your files will be present on the disk the next time you use the same machine, it will be pure coincidence if they are since other users may have used the same machine in the interim.

Since you have logged off, power down the machine. The video monitors are expensive, and have a finite lifetime.

4.0 Machine Locations and Availability

4.1 Machine Locations

As previously mentioned, the Department has the concepts currently configured in two LANs as follows:

Room	Machines	Accounts
ME318	10	USER1 - USER10
ME402w	5	SERC1 - SERC5

4.2 System Printer

Located in ME318 is the system printer connected to a microcomputer as a printer server. Spooled output via the vendor's SPOOL program and the Department's PRINT command (see section 4.4) is sent to this printer. The PRINT command includes a banner page as the first page of output, which contains the file name, time, and room of origin. The SPOOL command includes the user's account name, and file name on the banner page.

4.3 Availability

Unless the user is eligible to hold an access key and building pass, the machines are only available during regular working hours, or when arrangements have been made to have the laboratory open in the evening.

Undergraduate students should note that there is NO equipment available for obtaining letter quality hard copy output. The machines are not intended for use as word processors for undergraduate projects. Students will only be permitted to use the system line printer, using the standard paper provided, for text related to the CAD laboratory assignments.

5.0 The File System - FileMgr

5.1 Introduction

The file manager program is used to manipulate files (copy, delete, list). There are three types of files recognized by the file system: text files, data files, and temporary files. Text files have a ".TEXT" extension, while data files do not. Temporary files can be identified by "???" for the month and "100" for the year when the file is listed. File system commands only apply to non-temporary files; temporary files can only be deleted using a particular command. The file system is block structured, where files, (of which there can be at most 77 in any given volume), are allocated in volumes which are not unlike a floppy diskette. Due to the file systems block structured allocation scheme, it is periodically necessary to reclaim "holes" between files by "crunching" the volume according to the instructions below.

5.2 File References

A file reference has the following syntax:

```
{/volume | [/volume/]name}
```

The file name may be at most fifteen characters, including the ".TEXT" extension, if present. A volume reference, (which may be at most, seven characters), alone refers to a device (eg: /PRINTER), as opposed to a volume qualified file name (eg: /USER23/OUTPUT.TEXT).

5.3 Formatting Floppy Disks

function key for which the desired help is requested. This process is repeated until the user hits the ESC key.

US_F05	EdWord	Invokes the vendor supplied screen oriented editor. (See section below.)
US_F06	FileMgr	Invokes the file management program. (See section below.)
US_F07	WindowMgr	Invokes the window management program which is used to load display and keyboard character sets.
US_F08	ExecFile	Invokes the script file program, which takes as it's input a file containing operating system commands and program names which are to be executed in the order in which they are encountered in the file.
US_F09	Inst II	Invokes the system management software.
US_F10	SysUtils	Invokes the system utilities software which is used to load and initialize device drivers, set the time and date, and list the device drivers currently loaded.
US_F06	RevBkgnd	Sets the background colour of the currently selected window to the reverse of what it previously was (ie. if it was black, it will be changed to white).
US_F07	CreateWindow	Is used to create a user defined window. The user will be prompted for the upper left and lower right coordinates of the window, which are specified by moving the cursor via the arrow keys. If the window was successfully created, then it's contents will be erased and the window will be selected as the active window. A maximum of seventeen windows may be defined.
US_F09	MountMgr	Invokes the mount manager program which is used to mount and unmount volumes which the user has access to.
CSH_F07	TempWindow	Creates a temporary window in the same manner as described for create window, but the window is not erased when selected. Once another window is selected, the temporary window will be deleted.
CSH_F10	Reboot	Initiates a software reboot which reinitializes the machine to the state it was in when the user first logged on. This is a useful way of unloading any user loaded device drivers.

3.4 Log Off

The user may log off using three methods:

1. Turn off the power to both the base unit, and the display unit, or
2. Type LOGOFF, or
3. Enter Ctrl-BREAK by holding down the <CTRL> key and then hitting the <BREAK> key.

Methods one and three are appropriate for aborting 'run away' programs, (ie. programs which are apparently not working correctly).

If you are terminating your work session, then before departing ensure that you have retained copies of files that you may require at a later time, (see section 5 - 'The File System' for the procedures for doing this). There is

files, etc.).

- UTIL - Vendor supplied system utilities / system management utilities, etc.).
- USERnn - 1500 blocks for logon session files. This space is dedicated to machine nn only. Files created in this volume will remain for the duration of the logon session. Files remaining from previous users should be deleted.
- TEMP - 512 block temporary storage area for floppy disk file manipulation. This volume is equally accessible by all machines on the network, and is used as a means of passing your files from volume USERnn to your floppy disk, as the floppy disk drive cannot be accessed by a machine not directly connected to it.
- TILS - Carleton installed utilities and programs.
- ISYS - Vendor supplied integrated system programs including a communications program.
- CHARSET - Vendor supplied character set and printer setup files.

3.3 Function Keys

The operating system provides a convenient user interface via programmable function keys, essentially eliminating the requirement for written documentation and for entering commands via the keyboard. The user simply selects the desired function key which instructs the operating system to execute the desired function which could in fact result in another program executing. This virtually eliminates the learning and remembering commands - if you can't see the desired command on one of the function keys, then it probably doesn't exist. The operating system has provision for managing forty function key values: the display set, the shift display set, the command set, and the command shift set. Once successful logon has been accomplished, a set of function key labels in reverse video, (F1 to F10), will be displayed at the bottom of the screen. Each label corresponds to the respective function key on the top row of the key board. Some labels have two entries. The top entry is selected by holding down the shift key and then pressing the appropriate function key.

F1	F2	F3	F4	F5
		EdWord		.
p-System	LogiCalc	ISYS	SetVol	ListVol

F6	F7	F8	F9	F10
				.
ClrWindow	SelWindow	Restart	Help	

By pressing and holding the COMMAND key on the keyboard, the function keys will change to the following values:

For example, the user may execute a software reboot by hitting the F9 key, while holding down both the COMMAND and SHIFT keys.

The following table documents the various function key labels.

Function	Invocation Specification
US_Fnn	<Fnn>
SH_Fnn	<SHIFT> <Fnn>
CS_Fnn	<COMMAND> <Fnn>
CSH_Fnn	<COMMAND> <SHIFT> <Fnn>

Function	Name	Purpose
US_F01	p-System	Invokes the p-machine which is a popular computing environment developed by the University of California. Prospective users should consult the system specialist.
US_F02	LogiCalc	Invokes the spread sheet program based on the popular VisiCalc program.
US_F03	LSYS	Invokes the vendor's integrated system product which is aimed at 'office automation'. This environment includes programs for communication, spreadsheets, word processing, two and three dimensional graphics, data manipulation, desk calculator, and world clock.
US_F04	SetVol	Sets the default volume to which all unqualified file references will apply. The currently selected volume is displayed at the top centre of the screen, beside the user's account name.
US_F05	ListVol	Lists the contents of the user specified volume. If no volume name is provided, then the names of the files in the default volume are displayed, (simply hit carriage return for the volume name).
US_F06	ClrWindow	Clears (erases) the currently selected window (the one with the white border around it).
US_F07	SelWindow	Selects a user defined window. The Concept initially has two window defined - the root window, and the command window. The root window occupies the upper area of the screen, while the command window extends across the bottom of the screen just above the function key labels. Users may define up to twenty windows. The select window function selects the next defined window in a circular fashion.
US_F08	Restart	Submits the last command executed to the operating system for re-execution.
US_F09	Help	Initializes the help system to provide help for the various function keys. The user presses the

		function key for which the desired help is requested. This process is repeated until the user hits the ESC key.
SH_F03	Edword	Invokes the vendor supplied screen oriented editor. (See section below.)
IS_F01	FileMgr	Invokes the file management program. (See section below.)
IS_F02	WindowMgr	Invokes the window management program which is used to load display and keyboard character sets.
IS_F03	ExecFile	Invokes the script file program, which takes as it's input a file containing operating system commands and program names which are to be executed in the order in which they are encountered in the file.
IS_F04	Inst II	Invokes the system management software.
IS_F05	svs/tlts	Invokes the system utilities software which is used to load and initialize device drivers, set the time and date, and list the device drivers currently loaded.
IS_F06	RevBkgnd	Sets the background colour of the currently selected window to the reverse of what it previously was (ie. if it was black, it will be changed to white).
IS_F07	CreWindow	Is used to create a user defined window. The user will be prompted for the upper left and lower right coordinates of the window, which are specified by moving the cursor via the arrow keys. If the window was successfully created, then it's contents will be erased and the window will be selected as the active window. A maximum of seventeen windows may be defined.
IS_F09	MountMgr	Invokes the mount manager program which is used to mount and unmount volumes which the user has access to.
CSH_F07	TempWindow	Creates a temporary window in the same manner as described for create window, but the window is not erased when selected. Once another window is selected, the temporary window will be deleted.
CSH_F10	Reboot	Initiates a software reboot which reinitializes the machine to the state it was in when the user first logged on. This is a useful way of unloading any user loaded device drivers.

3.4 Log Off

The user may log off using three methods:

1. Turn off the power to both the base unit, and the display unit, or
2. Type LOGOFF, or
3. Enter Ctrl-BREAK by holding down the <CTRL> key and then hitting the <BREAK> key.

Methods one and three are appropriate for aborting 'run away' programs, (ie. programs which are apparently not working correctly).

If you are terminating your work session, then before departing ensure that you have retained copies of files that you may require at a later time, (see section 5 - 'The File System' for the procedures for doing this). There is

I guarantee that your files will be present on the disk the next time you use the same machine, it will be pure coincidence if they are since other users may have used the same machine in the interim.

Once you have logged off, power down the machine. The video monitors are expensive, and have a finite lifetime.

4.0 Machine Locations and Availability

4.1 Machine Locations

As previously mentioned, the Department has the concepts currently outlined in the FANS as follows:

<u>Room</u>	<u>Machines</u>	<u>Accounts</u>
ME318		SER1 - SER1
ME426		SER2 - SER2

4.2 System Printer

Located in ME318 is the system printer connected to a microcomputer as a printer server. Spooled output via the vendor's SPOOL program and the Department's PRINT command (see section 4.4) is sent to this printer. The PRINT command includes a banner page as the first page of output, which contains the file name, time, and room of origin. The SPOOL command includes the user's account name, and file name on the banner page.

4.3 Availability

Unless the user is eligible to hold an access key and building pass, the machines are only available during regular working hours, or when arrangements have been made to have the laboratory open in the evening.

Undergraduate students should note that there is **NO equipment available** for obtaining letter quality hard copy output. The machines are not intended for use as word processors for undergraduate projects. Students will only be permitted to use the system line printer, using the standard paper provided, for text related to the CAD laboratory assignments.

5.0 The File System - FileMgr

5.1 Introduction

The file manager program is used to manipulate files (copy, delete, list). There are three types of files recognized by the file system: text files, data files, and temporary files. Text files have a ".TEXT" extension, while data files do not. Temporary files can be identified by "???" for the month and "100" for the year when the file is listed. File system commands only apply to non-temporary files; temporary files can only be deleted using a particular command. The file system is block structured, where files, (of which there can be at most 77 in any given volume), are allocated in volumes which are not unlike a floppy diskette. Due to the file systems block structured allocation scheme, it is periodically necessary to reclaim "holes" between files by "crunching" the volume according to the instructions below.

5.2 File References

A file reference has the following syntax:

```
{/volume | [/volume/]name}
```

The file name may be at most fifteen characters, including the ".TEXT" extension, if present. A volume reference, (which may be at most, seven characters), alone refers to a device (eg: /PRINTER), as opposed to a volume qualified file name (eg: /USER23/OUTPUT.TEXT).

5.3 Formatting Floppy Disks

Temporary users, (those that do not have personal accounts), can retain copies of any files they create by storing them on floppy disks (eight inch, single sided, and single density). Before a floppy disk can be used for the first time on the Corvus system, it must be formatted. To format a floppy disk, log on to the Concept which has the floppy disk drive attached to it. Make sure that your floppy disk has a write tab over the write protect slot before attempting to format it. Turn on the disk drive, (the switch is on the back), and insert your disk, with the label on the lever side of the drive. Close the lever and type FFORMAT at which point the program should ask you which unit number - you enter the digit 9. It will then ask you if you want the disk to be initialized only - you enter the character N. The last question it will ask you is if you want to verify the format - you enter the character N, (otherwise it will take approximately ten times longer to initialize the floppy disk). The formatting will then commence (you should hear a periodic series of noises originating from somewhere inside the drive, and the red disk light should be illuminated). If, instead, you get an error message, check to ensure that the disk is properly inserted in the drive, that the drive is powered on, and that the write protect slot is covered by an opaque tab (a piece of tape is not sufficient). Once the formatting has successfully completed, your floppy disk will always have a volume name of /FLOPPY which must be used when transferring to/from the floppy. NEVER power a Concept on or off with a floppy disk in the drive - everything stored on the disk will probably become inaccessible. It is always a good practice to retain more than one copy of any valuable file on separate floppy disks.

5.4 Copying Files

A file or files may be copied from one volume to another using the copyfile command. Once the command has been selected, the program will prompt you for the name of the file(s) to be copied. Enter the file name(s) to be transferred. The program will then ask you for the name of the destination volume. Enter the volume name with the leading slash (ie. /TEMP). Providing that the file(s) exist, that you have access to the destination volume, and that there is space in the destination volume, the program will transfer the file(s). You may also use an asterisk to indicate wild fields. In our environment, we can not provide all Concepts with floppy disk drives, so we have adopted the following procedure. Only one machine in the CAD lab is equipped with a floppy disk drive. Users wishing to transfer information into or out of the system do so by copying the desired files from their USERnn volume into the /TEMP volume. This makes the files accessible to the machine which has the floppy disk drive connected to it. For example, if a user is beginning a work session station 25, then the work files are copied from the floppy disk into the /TEMP volume, and then the user returns to the USER25 station and copies the work files into the /USER25 volume. The reverse is followed when saving a file on floppy - copy the work files into the /TEMP volume, and then go to the machine with the floppy drive attached, and copy the files from the /TEMP volume to the /FLOPPY volume (make sure that the floppy is in the disk drive).

Example:

To copy all text files in /TEMP to /FLOPPY:

CopyFile Copy which file(s)? /TEMP/*.TEXT

CopyFile Copy file(s) to which volume? /FLOPPY

5.5 Deleting Files

Files are deleted by using the DelFile function and entering the possibly wildcarded filenames at the prompt. (Temporary files are deleted by using the DletTemp function.)

Example:

1. To delete all files with the prefix cif_ in the default volume:

DletFile Delete which file(s)? cif_*

2. To delete all the temporary files in the /TEMP volume:

DletTemp Delete which temporary file(s)? *

5.6 Listing Volume Contents

Use the ListVol command to list the names, sizes, and date last modified, of the files in the selected volume. If you want list the files in the default volume, enter a carriage return for the volume to be listed, otherwise specify the name of the volume to be listed.

5.7 Volume Crunching

Periodically, because of the manner in which the file system was implemented, it is necessary to reclaim the unusable space in a volume (for example: there might be a total of ten unused blocks in a volume, where the largest gap is three blocks. In this case, the largest file that could be created is three blocks.) Crunching a volume will coalesce all the unused disk space to the end of the volume. The Crunch command is used to reclaim disk space.

Example:

To reclaim all the unused disked space in the /TEMP volume:

Crunch Crunch which volume? /TEMP

The program will indicate the names of the files being moved to the top of the volume.

5.8 Further Information

See the Personal Workstation User Guide for more information.

6.0 System Editors

Two editors are available for use on the Concept. EdWord, the vendor supplied product, is a window editor suitable for users working with large files (up to 100K characters). Use of this editor is discouraged because it requires more time to learn, and it takes a significant length of time to enter and exit. E, an editor written at Carleton, is a context oriented line editor which is very similar to the one offered on the University's mainframe. One important enhancement is macros which automate sequences of edit commands. There is a default macro file on entry, to which the user can add his own macros.

6.1 E - A Line Oriented Editor

This is a context oriented line editor with character, record and macro commands. Written by Peter Choynowsky for the Department of Electronics, this editor closely resembles the editor available on the Honeywell Level 66 machine, offering a fast response rate suitable for a workstation environment. This editor possesses a healthy subset of the commands available, and any behavioral differences are documented in the editors interactive help facility.

To invoke the editor type:

SysLevel Select function: E [filename]

Online help is available once the editor is entered by typing:

* HELP TOPICS

Copies of the editor help file and the default macro file are provided in one of the attached appendices.

6.2 EdWord - A Window Editor

EdWord, in conjunction with the vendor supplied tutorial file adequately acquaints the user with the editing commands. It is important to realize that information accessible while inside the editor is not accessible outside

sets.

To create a rectangle, the user moves the dynamic graphic cursor, (the cross hair), via the keyboard cursor keys, to one of the desired corners. The `<Slew>` command then places the static graphic cursor, (the diamond), at this coordinate, permitting the user to then position the dynamic cursor at the opposite corner of the desired rectangle. The `<Mark>` command actually creates the rectangle on the currently specified level, (in the case of the screen layout above, a 10 by 10 lambda rectangle would be placed on the P-well layer centered about the coordinate (35,5)). Graphical objects are deleted in a similar manner by using the `<Delete>` command instead of the `<Mark>` command, again, on the currently selected layer. During deletions, rather than redrawing the screen each time a rectangle is deleted, the deleted object is visually marked as being deleted by drawing a cross through it. The user may explicitly update the screen to remove the rectangles marked for deletion by using the `<Redraw>` command. The cursor status information is constantly updated to reflect the cursor coordinates of the dynamic cursor, and the absolute difference between the dynamic and static cursors, in dimensionless units. As the cursor keys are set up to report linear incremental movement, the user can select between slow slewing, (the `<SlewSlow>` command), and faster slewing, (the `<SlewFast>` command). Initially, the slow slew rate is set to one lambda, and the fast slew rate to ten lambda. However, the user can adjust these rates by using the appropriate `<Set SlowSlew>` and `<Set FastSlew>` commands. Currently, the layout space is defined to be (0.4800, 0.4800) lambda. Using one CIF unit to equal one micron will require a 100/2 scaling in the DS statement to achieve the appropriate real world units. In the fast slew mode, the graphical cursor is relocated to the nearest integral multiple of the fast slew rate. If the user has the grid turned on, and at a grid spacing corresponding to the fast slew rate, then the cursor will move on the grid points. As with the cursor movement speeds, the user can specify the grid spacing via the `<Set Grid>` command. Finally, the user can set the overall display magnification.

As single bit pixel graphics are used, rectangles on the currently active layer are displayed using dotted lines, while all other visible objects are displayed using solid lines. The user is able to select a subset of the complete layer set for display by using the `<Command-Set Visible>` command. The status window located above the function key annotation area reflects this change by indicating in reverse video, the layers which are currently visible, (invisible layers are indicated in non-inverse video). The currently active layer is selected by the `<Set Current>` command, and the status area will be updated by indicating the current layer over the visible set status line to remind the user whether or not that layer is currently visible. When the moving cursor encounters a screen boundary, the screen is updated in the direction of travel by a pre-selected screen update fraction, (which is established by using the `<Command-Set ScrnSlew>` command). The complete layout window is redrawn as it is currently faster to do so than scroll the video memory by the desired fraction.

Hard copy output is available by using a HP7221C eight pen plotter connected to DtaCom1 via a null modem on the Corvus side, using X-On/X-Off protocol at 2400 baud. The program automatically loads the data comm driver.

<u>Function</u>	<u>Purpose</u>
Cursor Keys	- Move the dynamic, (cross hair), cursor about the layout space, at the currently selected slew rate (see <code>SlewSlow</code> and <code>SlewFast</code>).
Slew	- Moves the static cursor, (the diamond), to the coordinates of the dynamic cursor (the cross hairs).
Mark	- Creates a rectangle on the currently selected layer, between then static and dynamic coordinates, which is drawn using dotted lines. No attempt is made to avoid coincident primitives occurring on the same layer. The program will only create right rectangles, (ie. oriented parallel to the principle axes), and there is no restriction as to the location of the static and dynamic

the editor until the file is explicitly saved. Also, information accessible from inside the editor is not lost if the user leaves the editor. This will only happen if the EdWord workspace file is deleted. When EdWord is invoked, it will ask you for the name of the workspace file (the default is W). If the workspace file does not exist, then the program will ask you if you wish to create one, and the number of blocks (default is 494). The defaults are quite adequate for most work. EdWord will then read in a tutorial file which may be used to introduce you to the editor. Before a simulation file can be created, you need to either clear out the default, (W:HELLO), workpad, or create a new one using the WorkPad option. Once a simulation file has been created inside the editor, it must be saved outside the editor using the WorkPad SaveFile option, using the filename INPUT. The exit the editor and invoke the desired simulator. A complete description of the editor can be found in the Personal Workstation User Guide, though the tutorial is satisfactory for most purposes.

7.0 Using CAD Tools

7.1 IC Layout

Over the past two years, the Department has developed a program (CIFDRAW) to permit users to graphically lay out integrated circuit designs. The program is based on the Manhattan system, where all graphical objects are rectangles oriented parallel to the X axis. The program currently creates a monolithic Caltech Intermediate Form (CIF) symbol, which may be a complete design, or a part of a larger design.

After the program has completed some preliminary initialization, the user is presented with the following video display:

Cursor X -	30	Y -	0	Current	PW
Delta X -	10	Y -	10	Visible	PW F PG NG CP P NP PP C M G

F1	F2	F3	F4	F5	
Delete Mark	% Effic	Redraw		Set Current	<Shift> <Unshift>
Set Visible	ZERO Plot	Set Mag	Set SlowSlew	Set FastSlew	<Command-Shift> <Command-Unshift>

F6	F7	F8	F9	F10	
Grid Off Grid On	Goto	SlewSlow	SlewFast	Slew	<Shift> <Unshift>
Set ScrnSlew	Set Grid ListFile	SaveFile Get File	Or Line Xor Line	>Dump< Exit	<Command-Shift> <Command-Unshift>

The function key labels are organized into two groups. The primary label set, (the ones not preceded by Command-), is the default display. The secondary set replace the primary set when the keyboard <COMMAND> key is depressed. Many of the above commands lead to secondary function key label

sets.

To create a rectangle, the user moves the dynamic graphic cursor, (the cross hair), via the keyboard cursor keys, to one of the desired corners. The <Slew> command then places the static graphic cursor, (the diamond), at this coordinate, permitting the user to then position the dynamic cursor at the opposite corner of the desired rectangle. The <Mark> command actually creates the rectangle on the currently specified level, (in the case of the screen layout above, a 10 by 10 lambda rectangle would be placed on the P-Well layer centered about the coordinate {35,5}). Graphical objects are deleted in a similar manner by using the <Delete> command instead of the <Mark> command, again, on the currently selected layer. During deletions, rather than redrawing the screen each time a rectangle is deleted, the deleted object is visually marked as being deleted by drawing a cross through it. The user may explicitly update the screen to remove the rectangles marked for deletion by using the <Redraw> command. The cursor status information is constantly updated to reflect the cursor coordinates of the dynamic cursor, and the absolute difference between the dynamic and static cursors, in dimensionless units. As the cursor keys are set up to report linear incremental movement, the user can select between slow slewing, (the <SlewSlow> command), and faster slewing, (the <SlewFast> command). Initially, the slow slew rate is set to one lambda, and the fast slew rate to ten lambda. However, the user can adjust these rates by using the appropriate <Set SlowSlew> and <Set FastSlew> commands. Currently, the layout space is defined to be {0.4800, 0.4800} lambda. Using one CIF unit to equal one micron will require a 100/2 scaling in the DS statement to achieve the appropriate real world units. In the fast slew mode, the graphical cursor is relocated to the nearest integral multiple of the fast slew rate. If the user has the grid turned on, and at a grid spacing corresponding to the fast slew rate, then the cursor will move on the grid points. As with the cursor movement speeds, the user can specify the grid spacing via the <Set Grid> command. Finally, the user can set the overall display magnification.

As single bit pixel graphics are used, rectangles on the currently active layer are displayed using dotted lines, while all other visible objects are displayed using solid lines. The user is able to select a subset of the complete layer set for display by using the <Command-Set Visible> command. The status window located above the function key annotation area reflects this change by indicating in reverse video, the layers which are currently visible, (invisible layers are indicated in non-inverse video). The currently active layer is selected by the <Set Current> command, and the status area will be updated by indicating the current layer over the visible set status line to remind the user whether or not that layer is currently visible. When the moving cursor encounters a screen boundary, the screen is updated in the direction of travel by a pre-selected screen update fraction, (which is established by using the <Command-Set ScreenSlew> command). The complete layout window is redrawn as it is currently faster to do so than scroll the video memory by the desired fraction.

Hard copy output is available by using a HP7221C eight pen plotter connected to DataCom1 via a null modem on the Corvus side, using X-On/X-Off protocol at 2400 baud. The program automatically loads the data comm driver.

<u>Function</u>	<u>Purpose</u>
Cursor Keys	- Move the dynamic, (cross hair), cursor about the layout space, at the currently selected slew rate (see SlewSlow and SlewFast).
Slew	- Moves the static cursor, (the diamond), to the coordinates of the dynamic cursor (the cross hairs).
Mark	- Creates a rectangle on the currently selected layer, between then static and dynamic coordinates, which is drawn using dotted lines. No attempt is made to avoid coincident primitives occurring on the same layer. The program will only create right rectangles, (ie. oriented parallel to the principle axes), and there is no restriction as to the location of the static and dynamic

cursor locations except that they must be diagonal to one another. The bell will ring if the user attempts to create a zero width or height rectangle.

- Delete - Deletes the rectangle bounded by the static and dynamic cursors on the currently selected layer if it exists. Rather than update the screen each time a rectangle is deleted, a cross is drawn through it. The user can use the redraw command to update the screen.
- Goto - Prompts the user for the coordinate to which the dynamic cursor is to be moved to.
- SlewSlow - Sets the current stepping rate of the dynamic cursor to the slow slewing rate (which is set initially to one cif unit). The slow stepping rate may be altered by using the SlewSlow command.
- SlewFast - Sets the current stepping rate of the dynamic cursor to the fast slewing rate (which is initially set to ten cif units). The fast stepping rate may be altered by using the Set SlewFast.
- SetCurrent - Sets the currently active layer. Any rectangles created after this command has executed will be instantiated on the current layer. The status information will be updated to reflect the change. All rectangles created on the currently active layer are drawn using dotted lines, while all others are drawn using solid lines.
- SetVisible - Sets the set of currently visible layers. All visible layers are indicated in reverse video in the status area, while currently invisible layers are indicated in normal video. When this option is selected, the label area indicates +layer name, indicating that the corresponding function key will render that layer as visible. If the command key is pressed, then the labels will change to -layer name, indicating that the corresponding function key will render that layer as being invisible. This function is useful in design checking as it allows the designer to probe at various mask levels.
- Grid On - Turns on the layout space grid marks. The grid is automatically turned off if a magnification of 1/4 or less is selected using the Set Mag command. Initially, the grid spacing is set to one dot every ten cif units. This value may be changed by using the set Grid command.
- Grid Off - Turns off the layout space grid marks.
- Redraw - Redraws the currently visible portion of the layout space.
- Plot - Plots a user specified portion of the layout space on a locally attached HP7221C eight pen plotter attached to data comm 1 via a null modem at the Concept end. The user is prompted for the lower left and upper right coordinates of the area to be plotted. Only geometries on the currently visible layers are drawn. The user can specify the desired page size to be used. If the desired layout space to be plotted will not fit on one sheet of paper, the program will automatically paginate the plot, to be later cut and pasted together using the alignment marks drawn on the paper. During multiple page plots, the program halts after each page is plotted, to allow the user to change the paper. Once this has occurred, the user hits Continue to draw the next sheet, or Stop to quit.

- Zero - Erases all rectangles from the currently visible set of layers. To protect the user, this command temporarily sets all layers as being invisible, and then prompts the user to indicate which layers are to be visible for the zeroing.
- Set SlowSlew - Sets the stepping rate that is to be used when the SlowSlew command modifies the speed at which the dynamic cursor is to move at. This command specifies the movement speed in terms of cif units per arrow key hit.
- Set FastSlew - Sets the stepping rate that is to be used when the FastSlew command modifies the speed at which the dynamic cursor is to move at. This command specifies the movement speed in terms of cif units per arrow key hit.
- Set ScrnSlew - Sets the portion of the screen which is to be updated when a screen boundary is reached when moving the dynamic cursor.
- Set Mag - Sets the layout screen magnification. When the program is invoked initially, the magnification is said to be one. From this point, the magnification can then be increased or decreased. If the magnification is less than 1/4, the grid is automatically turned off.
- Set Grid - Sets the grid spacing in units of cif.
- SaveFile - Saves the complete layout space as a single cif symbol. It is suggested that the user use a .TEXT extension on the file.
- Get File - Reads a previously saved cif file, (which is a symbol), into the layout space, using the coordinate of the dynamic cursor as a local origin. This allows the designer to save a symbol, then instantiate it throughout the layout space.
- ListFile - List the contents of the user's directory. This is useful for determining if the layout space has been saved.
- Exit - Is used to leave the program. If the layout space has not been saved, then the program will prompt the user for a file name to save the design in. If you don't want to save the design, then just hit the carriage return.

A utility program, (CIFBIAS), has been provided to rotate and reflect CIF symbols. The user may in effect create a standard cell definition, and then transform copies of this symbol for later use. The program is self documenting, and may be found in the /UTILS volume.

7.2 Simulation

The Department has converted two simulation programs from the University of California - Berkeley to run on the Corvus computers. SPICE is used for analog simulation while SPLICE is used for digital and mixed mode simulation.

7.2.1 "INPUT.TEXT" File

The Department has automated the edit - simulate loop. Unless the user is doing something unusual, then the simulators may be invoked in a standard fashion, where the input will be taken from the file INPUT.TEXT in the user's currently selected volume. If this is not the case, then the user can use the more general approach, where the program(s) required to execute the simulation are individually invoked, using the user specified input files. Both approaches are described under the appropriate section.

7.2.2 Analog Simulation

The Department has converted SPICE 2G.6 to run on the Corvus computers. This general-purpose time and frequency domain circuit simulation program can be used for nonlinear dc, nonlinear transient, and linear ac analysis of circuits composed of resistors, capacitors, inductors, mutual inductors, independent voltage and current sources, four types of dependent sources, transmission lines, diodes, BJT's, JFET's and MOSFET's.

The SPICE User's Manual may be purchased in the Carleton Bookstore.

Users requiring only the MOS level 1 model can use the standard invocation. If the other semiconductor models are required, (including the MOS level 2 and 3 models), then the extended invocation is to be used.

Standard invocation:

SysLevel Select function: /UTILS/SPICERUN

SPICERUN is an executive program which deletes any temporary files before invoking SPICE. SPICE will read its input from the file INPUT.TEXT in the user's currently selected volume. This command must be issued from the SysLevel prompt, (if not, SPICE will issue an error message), ie. you cannot be at the file manager level when attempting to run SPICE. If there were no errors in the simulation, then the file PLOT.TEXT will contain the simulation results. Any temporary files and PLOT.TEXT are deleted before the run starts. The output file may be printed on the system line printer as described below.

Extended invocation:

Once per logon session, before the extended invocation can be utilized, the system stack pointer must be moved to permit more code space for the extra model code. This is done by:

SysLevel Select function: SP 0000

The extended invocation has the syntax:

SysLevel Select function: /UTILS/SPICE input_file output_file FULL

where: input_file - is the file containing the simulation definition.

output_file - is the file in which simulation results will be placed in. If the user wishes, the output from the simulation made be redirected instead to the console by substituting /CONSOLE for the output file.

FULL - indicates that all the models are to be made available. If there is not enough memory to load the required code, then a Loader 1 error message will be returned. Check to make sure that you have moved the stack pointer as described above, and that you are at the SysLevel prompt. Users should note that this form of invocation will substantially increase the amount of time required to perform the simulation as the program running in this mode is heavily overlaid to allow room for the complete model set.

7.2.3 Digital Simulation

SPLICE version 1.7 is a mixed-mode time domain simulation program designed to simulate systems composed of: (i) MOS devices, resistors and capacitors (electrical analysis), and (ii) logic gates (logic analysis). Both electrical and logic analysis are supported simultaneously, allowing the user the choice of the method, or mix of methods, most appropriate for the each part of the circuit being evaluated. For example, logic analysis may be used for portions of the circuit where first-order timing information

is sufficient, while electrical analysis can be used where detailed voltage waveform information is required.

SPLICE may be used to simulate circuits composed of the following components: (i) electrical: n-channel and p-channel transistors, drivers and loads, diodes, floating and grounded capacitors, resistors, voltage sources; (ii) logic: buffers, inverters, and, nand, or, nor, xor, nxor, nmos, pmos and cmos unidirectional and bidirectional pass gates; (iii) interface: voltage to logic and logic to voltage converters (which are used to interface electrical and logical circuits).

The SPLICE User's Manual is available from the Carleton Bookstore.

SPLICE is composed of three programs. The Berkeley Language Translator (BLT) is used to convert the user's input file into a form which SPLICE can ingest - a circuit network list. The expression "net list" is a common term in electronics CAD, and means an abstract list of connections of a circuit. Most logic simulators take such a net list. SPLICE is the actual simulation engine, while SPLOT is a post processor used to convert the unintelligible SPLICE output into plots and tables that the user can study.

Standard invocation:

SysLevel Select function: /UTILS/SPLICERUN

The extended invocation, as illustrated below, pictorially describes the functions that this executive program performs.

Extended invocation:

The diagram below illustrates how the various program which comprise SPLICE are linked together.

The following describes each program required to perform a SPLICE simulation, the invocation syntax, and examples:

1. BLT

Syntax:

- SysLevel Select function: /UTILS/BLT input output error

Examples:

SysLevel Select function: /UTILS/BLT input blt_out /consoleSysLevel Select function: /UTILS/BLT input blt_out errors

Description:

BLT takes the source (text) file (input) for a SPLICE simulation and generates the appropriate simulation engine input file, (the net list file). The input listing, any errors, and the circuit node map are stored in the error file. A prototype schematic entry program, (allowing the user to graphically define the circuit to be simulated), which creates a BLT compatible input file, is currently running, but not yet released.

2. SPLICE

Syntax:

SysLevel Select function: /UTILS/SPLICE input_file output_file

Example:

SysLevel Select function: /UTILS/SPLICE blt_out splice_out

Description:

SPLICE is the actual simulation engine, taking its input from the source file which was preprocessed by BLT (input file). Providing there were no errors, the simulation results will be stored in the output file.

3. SPLOT

Syntax:

SysLevel Select function: /UTILS/SPLOT input_file output_file

Example:

SysLevel Select function: /utils/splot splice_out splot_out

Description:

SPLOT post processes the SPLICE output, manipulating it into a form which the user can read, (creating a file with the .TEXT extension). Currently, this post processor has the ability to produce waveform and state transition diagrams. A prototype graphics post processor is running, but not yet released.

7.3 Typing Files

Files with the .TEXT extension may be displayed on the console by using the type command. The filename does not have to be qualified with the .TEXT extension for these commands to function.

Syntax:

SysLevel Select function: TYPE filename

Examples:

SysLevel Select function: type plot

SysLevel Select function: TYPE INPUT.TEXT

7.4 Printing Files

Hardcopy output of text files may be obtained by using either the vendor supplied SPOOL command, or the PRINT command. The print command differs from the SPOOL command in that it paginates the output, adding line numbers.

SPOOL Syntax:

SysLevel Select function: SPOOL filename

SPOOL Example:

SysLevel Select function: SPOOL plot

PRINT Syntax:

SysLevel Select function: PRINT filename

PRINT Examples:

SysLevel Select function: print plot

SysLevel Select function: PRINT INPUT.TEXT

Users should be cautioned about indiscriminately creating output. One system printer is shared between two networks. There is no printing priority, and occasions arise where a file spooled for printing after the user's file is actually printed first. Check to ensure that your listing is not attached to someone else's and that the system printer is actually operating. If it is not working, then locate a system specialist who will assist you. Also, remember that these networks are prohibited for undergraduate project word processing.

8.0 Conrus/Host File Transfers

Within the vendor supplied ISYS level is a data communications program which may be used for both terminal emulation, and file transfers to and from other machines. A Gandalf "Blue Box" must be locally attached, (the program cannot emit a break condition), to the Concepts DtaCom1 port. Also, from within the program, the ports baud rate and handshaking must be set to 2400 and XON/XOFF respectively. Facilities are available for completely handshaked large file exports, (information may be obtained from the system specialist).

9.0 System Features

The following section documents some of the lesser known system features.

9.1 Deleting Characters and Lines

At all system level software, the escape key, (ESC), may be used to erase the current input line.

The back space key, (BACKSPACE), deletes characters from the end of the line.

9.2 Input/Output Redirection

File input and output may be redirected from/to files in a way similar to Unix. Input from the standard unit may be obtained from a file as follows:

SysLevel Select function: myprog < mydata.text

Where myprog and mydata.text are the user's program and data files respectively. In the case of a Pascal program, all input from the standard file input will be satisfied from the file mydata.text.

File output may be redirected from the standard output to a file in a similar manner:

SysLevel Select function: myprog > results.data

9.3 Screen Scrolling

Screen output may be immediately halted by using <Ctrl S>, while <Ctrl Q> will restart the output. This is a useful feature when, for example, the user is viewing the contents of a file which have more lines than can fit on the screen.

9.4 Alias for /CCSYS

The /CCSYS volume has the character "!" as an alias. For example, /ccsys/paslib is synonymous to !paslib.

10.0 Using Language Processors

The Corvus Computer supports a popular variety of today's high level languages. Conveniently, Pascal, Fortran, and C all use the same code generator, which means that one can create programs with different parts written in different languages. With the exception of Basic and Lisp, all the compilers generate native 68000 code, which will allow the user to extract the maximum speed from the system, at the expense of compiling and linking. The source file containing the user's program is created using one of the previously described text editors, and depending on the language processor, is compiled, linked and executed, or simply interpreted.

10.1 Pascal

Invocation

To compile a pascal program:

SysLevel Select function: pascal program [-llisting] [-oobject]

program - the text file containing the Pascal source.

listing - the file that the optional program listing will be placed in

object - the new name of the object file. If this option is not specified, then the object code will be placed in a file with a name the same as the input file, with the extension ".OBJ".

To link a pascal program:

SysLevel Select function: linker program !paslib

To execute the program:

SysLevel Select function: program

Example

SysLevel Select function: pascal cifdraw -l/console

The above example will compile the file named cifdraw.text and generate a listing of the program on the user's terminal.

Documentation

CCC/34-33/1.1, February, 1982 "The Corvus Concept Pascal Reference Manual"

Comments

Users can call Fortran subroutines from within a Pascal procedure by declaring each routine to be called as an external procedure with var

parameters for non-array data, as all parameters to Fortran procedures are passed by reference, (the address of the object), as opposed to pass by value. Character strings cannot be passed to Fortran easily.

Example:

The user has a Fortran subroutine which performs an n point in-array, (the transformed data overwrites the original data), Fourier transform of the user supplied data.

```
Subroutine FFT (Reals, Imags, Points)
Real      Reals(1)
Real      Imags(1)
Integer   Points
```

In the Pascal program, before the procedure is invoked, the following must appear:

```
Type
Type_Data      =      Array [1..1024] Of Real;
```

```
Var
Real Points:      Type_Data;
Imaginary Points: Type_Data;
Number_Of_Points: LongInt;
```

```
Procedure FFT (Var Real Points, Imaginary Points: Type Points;
Var Number_Of_Points: LongInt); External;
```

The Fortran procedure is separately compiled, and all the procedures are linked. If the Pascal source is contained in the file SIGNAL.TEXT, and the Fortran FFT routine is contained in the file FFT.TEXT, then the following illustrates the compile and link steps:

1. Compile the Pascal source:

SysLevel Select function: pascal signal

If the program successfully compiles, then a file named SIGNAL.OBJ will be created in the current volume.

2. Compile the Fortran source:

SysLevel Select function: fortran fft

If the routine successfully compiles, then a file named FFT.OBJ will be created in the current volume.

3. Link the program:

SysLevel Select function: linker signal fft !ftnlib !paslib

If the link is successful, then a file named SIGNAL will be created in the current volume.

4. Execute the program:

SysLevel Select function: signal

10.2 Fortran

Invocation

To compile a fortran program:

SysLevel Select function: fortran program [-llisting] [-oobject]

program - the text file containing the Fortran source.

listing - the file that the optional program listing will be placed in.

object - the new name of the object file. If this option is not specified, then the object code will be placed in a file with the same name as the input file, with the extension ".OBJ".

To link a fortran program:

SysLevel Select function: linker program !ftnlb !paalib

To execute the program:

SysLevel Select function: program

Documentation

COO/32-66/1.1, February, 1982 "The Corvus Concept Fortran Reference Manual"

10.3 Other Languages

Various other languages including C, Basic, Lisp, and 68000 assembler are available. See the system specialist for further information.

11.0 Appendices

11.1 E - Help and Macro File Listings

*** Not Provided Here ***

END

1 5 0 4 8 6

FIN